

SÉRIE : A
N° D'ORDRE : 180
N° DE SÉRIE 21

THESE

présentée devant

L'UNIVERSITE DE RENNES

U. E. R. Mathématiques et Informatique

pour obtenir

Le grade de Docteur d'État ès-Sciences Mathématiques

par

Jean-Pierre VERJUS

Nature et composition des objets manipulés dans un
système de programmation.

Soutenue le 12 Septembre 1973 devant la Commission d'Examen

MM. M METIVIER

Président

L. BOLLIET

C. COATMELEC

C. PAIR

Examineurs



Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41553611_0024

SÉRIE : A

N° D'ORDRE : 180

N° DE SÉRIE 21

THESE

présentée devant

L'UNIVERSITE DE RENNES

U. E. R. Mathématiques et Informatique

pour obtenir

Le grade de Docteur d'État ès-Sciences Mathématiques

par

Jean-Pierre VERJUS

Nature et composition des objets manipulés dans un
système de programmation.

Soutenue le 12 Septembre 1973 devant la Commission d'Examen

MM. M METIVIER

Président

L. BOLLIET

C. COATMELEC

C. PAIR

Examineurs

SÉRIE A
N° D'ORDRE
N° DE SÉRIE

1873

L'UNIVERSITÉ DE LEIDEN

II E S. 1873-1874

Le grade de Docteur en Lettres



Nature of the work

1873-1874

Number of the work

MM. M. MESTRE

J. BOLLEET

C. GATMEIER

C. PAIR

UNIVERSITE DE RENNES

U.E.R. issues de la FACULTE DES SCIENCES

Doyens Honoraires

M. MILON Y.
M. TREHIN R.
M. SCHMITT M.
M. LE MOAL H.
M. MARTIN Y.
M. BOCLE J.

Professeurs Honoraires

M. FREYMANN R.
M. HAGENE P.
M. POISSON R.
M. ROHMER R.
M. TREHIN R.
M. SALMON-LEGAGNEUR F.
M. des ABBAYES H.
M. MILON Y.

Maîtres de Conférences Honoraires

M. GRILLET L.
M. LECRECQ A.

Mathématiques

Professeurs

Melle CHARPENTIER M.
M. GUERINDON J.
M. METIVIER M.
M. GIORGIUTTI I.
M. BRUNEL A.
M. TOUGERON J. C.
M. MIGNOT A.

Maîtres de Conférences & Chargés de Cours

M. LEGOUPIL J.
M. HOUDEBINE J.
M. EUVRARD D.
M. NEDELEC J. C.
M. VERJUS J. P.
M. TRILLING L.
M. FERRAND D.
M. GUIVARC'H Y.
M. KEANE M. (M. C. associé)
M. GILDELAMADRID (Prof. associé)

Physique

M. VACHER M.
M. VIGNERON L.
M. LE BOT J.
Mme ROBIN née SALOMOND
M. ROBIN S.
M. LE MONTAGNER S.
M. MEVEL J. Y.
M. MEINNEL J.
M. REGENSTREIF E.
M. BRUN P.
M. LE ROUX E.
M. DUBOST

M. GUIDINI
M. HAEUSLER C.
M. ARQUES P. Y.
M. DECAMPS E.
M. NUSIMOVICI M. A.

Chimie

M. VENE J.
M. LE MOAL H.
M. LEVAS E.
M. PELTIER D.
M. PRIGENT J.
M. JOUAN P.
M. FOUCAUD A.
M. LANG J.
M. CARRIE R.
M. GUERILLOT C.
M. KERFANTO M.
M. DABARD R.

M. GRANDJEAN D.
M. LUCAS J.
M. DUVAL J.
M. MARTIN G.

Zoologie

M. RICHARD G.
M. MAILLET P.
M. BOISTEL J.
M. RAZET P.
M. FOLLIOT R.
M. ALLEGRET P.
M. MANIEY J.
M. JOLY J.
M. POSTEL (Professeur associé)

M. LEFEUVRE J.C.

Botanique

M. VILLERET S.
M. CLAUSTRES G.
Melle GOAS M.
Melle GOAS G.

Mme LEMOINE C.
M. TOUFFET

Géologie

M. COGNE J.
Melle DURAND S.
M. PHILIPPOT A.

M. BOILLOT G.
M. HAMEURT J.

Le Secrétaire Général de l'Université

M. BOURDIGUEL R.

Je remercie Monsieur le Professeur METIVIER qui n'a cessé de m'encourager à rédiger cet ouvrage, et qui m'a fait l'honneur de présider ce jury.

Je remercie Monsieur le Professeur BOLLIET d'avoir dirigé mes premiers travaux, encouragé leurs prolongements, et accepté d'être le responsable scientifique de cette thèse.

Je remercie Monsieur le Professeur PAIR de l'intérêt qu'il a porté à mes travaux et de ses critiques constructives.

Je remercie Monsieur le Professeur COATMELEC de s'être intéressé à ce travail et d'avoir bien voulu participer à ce jury.

Le contenu de cette thèse est le résultat de l'expérience acquise de la réalisation du système DIAMAG à Grenoble, et de la réflexion entreprise avec Laurent TRILLING à Montréal, puis à Rennes. Aussi, j'exprime toute mon amitié et toute ma gratitude aux Grenoblois de DIAMAG, aux Montréalais et Rennais des équipes système, et tout particulièrement à Laurent TRILLING, compagnon de travail pendant ces cinq dernières années.

Je remercie Chantal et Madame LE BAIL qui se sont chargées de la réalisation matérielle de cette thèse.

TABLE DES MATIERES

	Pages
CHAPITRE 1. INTRODUCTION.	1
1.1. Point de vue sur la définition d'un système	1
1.2. Historique	2
1.3. Plan de l'ouvrage	5
Bibliographie	6
I - ETUDE DE LA NATURE ET DE LA COMPOSITION DES OBJETS MANIPULES DANS LES SYSTEMES DE PROGRAMMATION	11
CHAPITRE 2. OBJETS ET ACCES AUX OBJETS.	12
2.1. Introduction	12
2.2. Identificateur et désignation	15
2.3. Objets et relations entre ces objets	16
2.31. Objets élémentaires	16
2.32. Les procédures	17
2.33. Objets composés	17
2.4. Durée de vie d'un objet et d'une chaîne d'accès	20
2.5. Procédure et contexte	21
2.51. Objets manipulés par un processus	21
2.52. Lexique, contexte, état d'exécution	22
2.53. Lexiques, contextes et états d'exécution associés à une procédure	23
2.6. Fichiers. Processeurs. Modules	24
2.61. Les fichiers	24
2.62. Les processeurs	26
2.63. Les modules	27
Bibliographie *	28
CHAPITRE 3. CONTRAINTES APPORTEES PAR LE SYSTEME.	30
3.1. Introduction	30
3.2. Représentation des objets en mémoire	31
3.21. Représentation de la relation repérer	32

3.22. Notion de descripteur et de préfixe	32
3.23. Eléments d'un descripteur ou d'un préfixe	33
3.3. Conservation et partage des objets	34
3.31. Désignation des emplacements en mémoire	34
3.32. Partage des objets	36
3.33. Conservation des objets - Catalogue	38
Bibliographie	39
CHAPITRE 4. COMPILATION. EDITION DE LIEN.	40
4.1. Introduction	40
4.2. Etablissement de relations de désignation	40
4.3. Résolution du contexte. Edition de lien	42
4.31. Représentation d'un identificateur	44
4.32. Mise en oeuvre de la fonction de désignation. Edition de lien	45
4.4. Elaboration d'une valeur	46
4.5. Représentation d'une procédure	46
CHAPITRE 5. LE SYSTEME CLICS.	47
5.1. Introduction	47
5.2. La Mémoire segmentée	48
5.21. Désignation d'un segment par un processus. Le descriptif	48
5.22. Nom d'un mot de segment	49
5.23. Descriptif et processus	50
5.3. Langage machine et objets manipulés	50
5.31. Format des instructions	50
5.32. Les différents types d'objets manipulés	51
5.33. Problèmes liés à l'adressage des objets d'une procédure	53
5.4. Désignation des objets	54
5.41. Désignation des compodants de la procédure	54
5.42. Désignation des objets rémanents	55
5.43. Désignation des fichiers	55
5.44. Désignation des objets locaux	56
5.45. Désignation des paramètres effectifs	57
5.46. Illustration des mécanismes de désignation	58

5.5. Etablissement des relations de désignation	58
5.51. Objets internes, locaux et rémanents	60
5.52. Paramètres	61
5.53. Fichiers	61
5.54. Représentation des fichiers	65
5.55. Fonction d'accès à un fichier	66
5.6. Opérations sur les procédures	67
5.7. L'éditeur de lien	68
Bibliographie	69
 CHAPITRE 6. LE SYSTEME BURROUGHS B/6700	 70
6.1. Introduction	70
6.2. Le matériel	70
6.21. Notion de préfixe	70
6.22. Description des processeurs câblés	71
6.23. Les segments	72
6.3. Représentation des objets	73
6.31. Repères et valeurs élémentaires	73
6.32. Tableaux	73
6.33. Procédures, blocs et descriptif	74
6.4. Accès aux objets	75
6.41. Objets manipulés par une procédure	75
6.42. La zone d'évaluation	75
6.43. La région des locaux	75
6.44. Etat d'exécution à l'appel d'une procédure	76
6.45. La région des liens formel/effectif	76
6.46. Le contexte d'une procédure	77
6.47. Représentation d'une valeur de type procédure	79
6.5. Partage des objets entre un processus père et ses processus fils	81
6.51. Création de processus	81
6.52. Contexte commun à deux processus	82
6.53. Désignation des objets	83
6.54. Partage des objets programmes et des objets primitifs par une collection de processus	85
Bibliographie	86

II - CONTRIBUTIONS A LA DEFINITION DES OBJETS ET DE LEURS COMPOSITIONS DANS UN SYSTEME DE PROGRAMMATION	87
CHAPITRE 7. DIAMAG. OPERATIONS SUR LES CONTEXTES	88
7.1. Introduction	88
7.2. Conservation et partage de procédures	88
7.21. Conservation et partage de procédures dans DIAMAG	88
7.22. Généralisation	89
7.23. Conclusion	92
7.3. Utilisation des contextes à l'exécution. Interactivité	93
7.31. Utilisation des contextes dans DIAMAG	93
7.32. Généralisation	94
Bibliographie	95
CHAPITRE 8. NATURE ET COMPOSITION DES OBJETS CONSERVES DANS LE SYSTEME SAR	96
8.1. Introduction	96
8.2. Nature et gestion des objets définis dans un environnement ALGOL 68	97
8.21. Détermination des objets utilisables par un processus	98
8.22. Mise en relation de l'objet et rupture de la relation	99
8.23. Accès à l'objet	100
8.24. Création, destruction et conservation d'objets	103
8.25. Conclusion	103
8.3. Le catalogue des objets conservés	104
8.4. Nature, conservation et utilisation d'objets compilés	106
8.41. Le prolator	108
8.42. Conservation et utilisation d'objets compilés	110
8.5. Conclusions	113
Bibliographie	115
III - BIBLIOGRAPHIE GENERALE	117
IV - INDEX	122

CHAPITRE 1. INTRODUCTION.

1.1. POINT DE VUE SUR LA DEFINITION D'UN SYSTEME.

Ce document présente un ensemble de travaux concernant la définition et la mise en oeuvre de langages d'utilisation d'un système de programmation. Ce faisant, nous voulons montrer que la définition précise du langage d'utilisation est une phase essentielle dans la conception d'un système; elle est essentielle parce qu'un système est fait pour répondre aux besoins des utilisateurs; elle est aussi essentielle parce qu'elle doit précéder la phase de mise en oeuvre du système.

Deux points de vue s'opposent au sujet de la conception d'un système. Le premier considère un système comme défini par les machines virtuelles qu'il met à la disposition des utilisateurs. Le second considère un système comme un mécanisme de partage des ressources physiques d'un ordinateur; ce mécanisme, appelé le noyau, permet la mise en oeuvre des processeurs (compilateurs de langages, système de gestion de fichiers, ...) qui définissent alors ces machines virtuelles. (On peut consulter à ce propos les minutes d'un débat sur ce sujet dans [1], pp. 51-54.)

En ce qui nous concerne, nous soutenons le premier point de vue. Il y a à cela deux raisons, que nous présentons ci-dessous.

Il n'existe pas, à notre connaissance, de mécanismes universels d'allocation de ressources partagées. De tels mécanismes doivent être considérés comme des outils de mise en oeuvre, et donc choisis en fonction de la nature des machines virtuelles définissant le système. Pour se convaincre de ce point, il suffit d'observer la différence des techniques utilisées dans des systèmes comme MULTICS [2] et B/6700 [3].

Les utilisateurs échangent de l'information entre eux. Dans ce but, on doit définir des protocoles d'échange qui doivent être considérés au niveau du noyau du système. Un protocole d'échange étant un langage, nous pensons qu'il ne doit pas être défini indépendamment du langage d'utilisation. Ce problème

est analogue à celui des protocoles de liaison entre programmes écrits dans des langages différents.

1.2. HISTORIQUE.

Notre intention n'est pas de faire un mémoire de l'ensemble des travaux réalisés depuis la mise en oeuvre, à GRENOBLE, de systèmes conversationnels, jusqu'à la conception du système SAR, à RENNES. Nous consacrons simplement cette introduction à citer ces travaux afin d'une part de montrer comment nous sommes arrivés à la conviction énoncée ci-dessus, et d'autre part de situer les contributions que nous pensons avoir apportées dans le domaine de la définition des langages d'utilisation de systèmes.

A GRENOBLE (IMAG, 1965-1968), l'équipe système, "très Algoliste" sous l'influence de MM. BOLLIET et GASTINEL, avait tenu à donner au langage de commande du premier système DIAMAG [4, 5] la forme d'appels de procédures ALGOL 60. Ce n'était encore qu'un effort syntaxique. Parallèlement, l'étude des nouveaux calculateurs [6, 7] nous montrait l'existence de systèmes pléthoriques où les constructeurs rivalisaient entre eux pour offrir plus de "possibilités" aux utilisateurs; cependant, la justification de ces possibilités, appelées "fonctions", se basait sur des critères imprécis et le taux de redondance était certainement important.

Nous entreprîmes, à cette période, de définir un système simple, interactif, en axant nos efforts sur la définition d'un langage primitif de manipulation de fichiers, et sur celle d'un langage de communication entre un utilisateur et un sous-système conversationnel basé sur ALGOL 60 [8, 9, 10, 11, 12, 13]. Quoique nous ayons défini deux types de langages, manipulant des objets différents, nous avons circonscrit et défini complètement le langage d'utilisation avant d'aborder la mise en oeuvre du système. Nous étions aussi convaincus que le langage d'utilisation de système devait ne plus être le simple assemblage d'un langage de programmation et d'un langage dit de commande, mais en devenir la composition.

Une telle démarche présentait néanmoins un inconvénient, en ce sens qu'elle ne permettait que la construction de systèmes spécialisés, puisque chaque langage de programmation avait alors un caractère spécifique. Aussi convenait-il de disposer d'un langage universel autour duquel on puisse construire un système. Cette approche n'a de sens que si elle ne rejette pas l'utilisation des langages les plus répandus. C'est notre hypothèse qu'un schéma d'exécution sophistiqué, mis en oeuvre pour un langage universel, s'adapte mieux à l'exécution d'un langage comme FORTRAN que l'inverse. C'est aussi notre hypothèse qu'un langage universel est un langage dans lequel on peut exprimer des liaisons entre programmes. Une conséquence de ce point de vue est qu'il faut considérer le langage universel, non seulement comme le langage de programmation privilégié dans le système, mais aussi comme le langage à l'aide duquel toutes les fonctions "systèmes" doivent être exprimées, et enfin comme le langage de base de la machine. L'existence des calculateurs BURROUGHS [3] corroborent cette hypothèse, d'autant plus que le langage ALGOL 60 étendu qui en constitue la base n'a pas l'universalité désirée.

A MONTREAL (Université de MONTREAL, Département d'Informatique, 1968-1970), avec Laurent TRILLING, nous avons été convaincus que le langage ALGOL 68 [14] avait les propriétés souhaitées. Faisant équipe, nous entreprîmes de concevoir un système ALGOL 68 (Système SAM) et d'aborder sa conception sous trois angles:

- 1) expression de l'interactivité entre un utilisateur et son programme;
- 2) définition des relations entre un programme et son environnement: accès aux données, production de résultat, conservation et utilisation d'objets catalogués partageables;
- 3) expression, à l'aide du langage, de certaines fonctions qui nous semblaient essentielles d'un point de vue pratique, comme l'édition (au sens mise à jour).

Le premier point nous a conduit à généraliser les concepts mis en oeuvre dans DIAMAG. En premier lieu, un programme se construit progressivement [13] et non "incrémentiellement" (voir [15]); en second lieu, l'utilisateur communique avec son programme à l'aide du langage dans lequel ce dernier est écrit [13].

Le second point nous conduisit à chercher à exprimer, à l'aide du langage ALGOL 68, l'accès à des collections de données cataloguées, les fonctions d'entrées-sorties classiques étant fixées par le langage. Ayant choisi un

modèle [16], nous avons exprimé, en ALGOL 68, la structure des objets catalogués et les fonctions d'accès à ces objets. Le prologue de programmathèque du langage était le support naturel de définition de ces entités [17, 18].

Pour le troisième point, nous avons choisi de considérer un éditeur de texte comme un "objet composé" de procédures ALGOL 68 primitives manipulant des chaînes de caractères. (L'étude des fonctions de l'éditeur a quelque peu débordé le cadre du système SAM et a débouché sur une étude générale des propriétés d'un système d'édition de texte [19, 20]). Il nous fallait alors définir en ALGOL 68 un tel objet et en permettre la manipulation. Sa structure est la même que celle d'un programme ALGOL 68, en cours de construction; soit $\mathcal{P}$ cet objet. Son utilisation s'exprime par une suite d'appels de procédures de $\mathcal{P}$, soit $\mathcal{S}$, et est équivalente à l'exécution du programme: $\mathcal{P} \mathcal{S}$ <délimiteur de fin de programme>. Nous avons ainsi défini la notion de contexte: dans SAM, un contexte se présente comme un programme ALGOL 68 inachevé, répondant à certaines contraintes syntaxiques. Un contexte peut être construit interactivement et progressivement, puis conservé et réutilisé. Une procédure du prologue de programmathèque permet d'exprimer la commutation de contexte [17, 18].

Nous avons ainsi défini une version rudimentaire d'un noyau de système, décrite en ALGOL 68 [17, 18]. Pour en expérimenter rapidement la facilité d'utilisation, nous avons choisi alors de porter nos efforts sur la définition et la mise en oeuvre d'un sous-ensemble d'ALGOL 68, autour duquel construire le système. Cette mise en oeuvre a été réalisée sur le calculateur CDC 3100 du département d'Informatique [21]. Le système a été construit sur machine nue, à l'aide d'un langage du type PL/360 [22].

Notre premier travail en France (Université de RENNES, Département de Mathématiques et Informatique) fut de constituer et de former une équipe chargée de la réalisation sur un calculateur CII 10070 d'un compilateur ALGOL 68 et du système environnant, rebaptisé SAR. Nous avons cherché alors à approfondir nos connaissances sur le langage ALGOL 68 et sur les systèmes existants en participant aux travaux du groupe ALGOL 68 [23, 24, 25, 26] et d'un groupe de rédaction d'un ouvrage sur les systèmes [27]. Grâce aux travaux effectués dans le cadre de ce dernier groupe, nous avons pu confronter nos idées de conception de système avec les deux systèmes qui, à notre sens, ont le plus marqué ces

dernières années, les systèmes MULTICS et BURROUGHS B/6700 [27, 28]. En outre, des contacts fructueux nous ont permis de tirer parti de l'expérience acquise par certains groupes français dans la conception et la réalisation de systèmes (ESOPE, GEMEAUX).

Le système SAR, dont nous présentons certaines caractéristiques fondamentales dans le chapitre 8., n'est pas terminé à ce jour. Nous disposons d'une version expérimentale, utilisable seulement à l'aide du langage cible construit pour ALGOL 68. Il s'est révélé très positif que la mise en oeuvre, toujours parallèle à une réflexion qui s'en déduit et l'alimente, ait couvert une longue période de temps. C'est la raison pour laquelle nous ne fixons pas de délais stricts à la réalisation complète du système.

Les travaux concernant SAR et auxquels nous avons plus particulièrement contribué sont consignés en [29, 30, 31, 32, 33].

1.3. PLAN DE L'OUVRAGE.

Ce document se présente sous la forme de deux parties auxquelles s'adjoignent deux annexes non incluses.

La première partie est consacrée à l'exposé d'une terminologie destinée à préciser la nature et la composition des objets manipulés dans un système (chap. 2., 3., et 4.). Cette présentation est illustrée par une description rapide et précise de la nature et de l'utilisation des objets dans les systèmes CLICS (présentation simplifiée du système MULTICS) et B/6700 (chap. 5. et 6.).

La deuxième partie présente succinctement les contributions que nous pensons avoir apportées dans ce domaine. Elles concernent:

- la définition de procédures partageables, compilées indépendamment. Cette définition a été rendue possible grâce à l'expérience acquise dans DIAMAG [13] (chap. 7.).

- la définition de relations entre un programme ALGOL 68 et son environnement. Les relations précisées sont volontairement laissées dans le vague dans le rapport ALGOL 68 [24, 26]. (chap. 8.)
- la définition d'un nouveau concept, extension du bloc en ALGOL 68, appelé prolator. Grâce à cet instrument, il nous est possible de préciser la nature et l'utilisation de certains objets manipulés dans les systèmes [30]. (chap. 8.)

Deux articles [26, 30] traitant en détail ces deux aspects doivent être considérés comme des annexes à ce dernier chapitre.

Chaque chapitre est suivi d'une bibliographie, où les ouvrages sont ordonnés dans l'ordre de leur citation. Une bibliographie générale est présentée en fin d'ouvrage, dans l'ordre alphabétique des noms d'auteurs.

BIBLIOGRAPHIE

- [1] HOARE C.A.R., PERROT R.H.
Operating Systems Techniques
Academic Press (1972)
- [2] CORBATO F.J., VISSOTSKY V.A.
Introduction and Overview of the MULTICS System
FJCC Proceedings (1965)
- [3] CREECH B.
Architecture of the B/6500
Infotech Report 2 (1971)
- [4] AUROUX A., BELLINO J.
Système en Temps Partagé 1401/7044
Monographie d'Informatique n° 2, DUNOD (1966)

[5] VERJUS J.P.

Description en ALGOL d'un Analyseur-Générateur Conversationnel pour un Sous-Ensemble d'ALGOL 60

Note Technique - IMAG (1966)

[6] AUROUX A., BOLLIET L., TASSART G., VERJUS J.P.

Etude Comparative de Nouveaux Matériels Orientés vers la Multi-programmation et le Partage de Temps

Note Technique - IMAG (1966)

[7] VERJUS J.P.

Méthode d'Adressage dans les Systèmes Multiprogrammés

Note Technique - IMAG (1965)

[8] AUROUX A., BELLINO J., BOLLIET L.

DIAMAG: A Multi-Access System for On Line ALGOL Programming

SJCC Proceedings (1967)

[9] BELLOT M., SIRET L., VERJUS J.P.

DIAMAG (II): Un Système Conversationnel à Accès Multiple

RIRO, 12.B2 (1968)

[10] SIRET L., VERJUS J.P.

DIAMAG II

Actes Séminaires IMAG (1968)

[11] SIRET Y., VERJUS J.P.

Description d'un Compilateur ALGOL Intégré dans un Système à Temps Partagé

Actes Congrès AFIRO (1967)

[12] VERJUS J.P.

Solutions Adoptées pour l'Ecriture d'un Générateur et d'un Interpréteur ALGOL Conversationnel

Actes Séminaires IMAG (1967)

[13] VERJUS J.P.

Etude et Réalisation d'un Système ALGOL Conversationnel
Thèse Doc-Ing , GRENOBLE (1968)

[14] VAN WIJNGAARDEN, MAILLOUX B.J., PECK J.E.L., KOSTER C.H.A.

Report on the Algorithmic Language ALGOL 68
Springer Verlag, BERLIN (1969)

[15] GRIFFITHS M., PECCOUD M., PELTIER M.

Incremental Compilation of ALGOL for a Multi-Access System
IMAG (1967)

[16] BARRON D.W., FRASER A.G.

File Handling at Cambridge University
SJCC (1967)

[17] TRILLING L., VERJUS J.P.

Définition d'un Système Extensible
Pub. 38 - MONTREAL (Dept-Info) (1970)

[18] TRILLING L., VERJUS J.P.

An Attempted Definition of An Extensible System,
in ALGOL 68 Implementation J.E.L. Peck (Ed.)
North Holland (1971)

[19] VERGES J.C.

Edition de Textes Basée sur la Reconnaissance Automatique de leurs Structures
Assises ACI (1972)

[20] VERGES J.C., VERJUS J.P.

Reconnaissance Automatique des Structures de Texte en vue de l'Edition
RAIRO B.3 (1973)

[21] HARLOUCHET C., RICHER N., VAUCHER J.

Le Système Extensible SAM
Pub. 79 - MONTREAL (Dept-Info) (1972)

[22] ASSABGUI M., MARCIL M., VERJUS J.P.

LP 3100 : Un Langage d'Assemblage Evolué

Pub. 42 - MONTREAL (Dept-Info) (1970)

[23] TRILLING L., VERJUS J.P.

Notions sur l'Implémentation d'ALGOL 68

Note Technique - RENNES (1971). A paraître dans [24]

[24] BACCHUS P., PAIR C., PECCOUD D. (Ed)

Manuel d'ALGOL 68 par le groupe ALGOL de l'AFCET

A paraître (Hermann)

[25] TRILLING L.

Traitement des Procédures

Journées ALGOL 68 - Institut Programmation PARIS (1973)

[26] VERJUS J.P.

Relation avec l'Environnement

Journées ALGOL 68 - Institut Programmation PARIS (1973)

[27] CROCUS (Ed)

Les Systèmes d'Exploitation des Calculateurs. Principes de Conception
par un Groupe de Rédaction

A paraître (Dunod)

[28] VERJUS J.P.

Quelques Propriétés des Langages d'Utilisation de Systèmes
Ecole d'été AFCET (1973)

[29] TRILLING L., VERJUS J.P.

Le Système SAR

Actes Séminaires IRIA (Structure et Programmation des Calculateurs) (1972)

[30] TRILLING L., VERJUS J.P.

Introduction Dynamique de Processeurs dans un Environnement Conversationnel
Article proposé pour publication (1973)

[31] ANDRE F., HERMAN D., KERANGUEVEN A., LE PORCHOU M.

Environnement Système pour ALGOL 68

Note Technique 010 - RENNES (1972)

[32] ANDRE F., HERMAN D.

Choix d'un Mécanisme d'Adressage Segmenté Réalisable sur Calculateur

CII 10070 et Adapté à un Système Basé sur ALGOL 68

Note Technique 014 - RENNES (1973)

[33] KERANGUEVEN A.

Implémentation et Gestion sur le Calculateur CII 10070 des Structures

Nécessaires à l'Exécution d'un Système Basé sur ALGOL 68

Concours ACM - PARIS (1973)

Première Partie

ETUDE DE LA NATURE ET DE LA COMPOSITION DES OBJETS MANIPULES DANS LES SYSTEMES DE PROGRAMMATION

Chapitre 2. Objets et Accès aux Objets

Chapitre 3. Contraintes apportées par le Système

Chapitre 4. Compilation. Edition de lien

Chapitre 5. Le Système CLICS

Chapitre 6. Le Système BURROUGHS B/6700

CHAPITRE 2. OBJETS ET ACCES AUX OBJETS.

2.1. INTRODUCTION.

Nous considérons l'information manipulée dans un système informatique comme un ensemble d'objets liés par des relations dont nous allons préciser la nature.

Les objets physiques. Les objets physiques sont les constituants d'un calculateur. Parmi eux, certains sont appelés emplacements (un mot, un byte). A un emplacement est attaché un nom que l'on peut assimiler provisoirement à une adresse. L'état des digits binaires qui composent un emplacement est appelé son contenu; un contenu a donc un caractère variable dans le temps. Enfin, une machine est dotée de processeurs élémentaires, les instructions, que nous appelons procédures technologiques.

Les objets informatiques. Les utilisateurs d'un système peuvent manipuler des objets informatiques [1] définis à l'aide de langages de programmation ou de commande. Lorsque le langage est celui de la machine, les objets informatiques sont les objets physiques de la machine. Lorsque le langage est "évolué", il permet de définir des objets informatiques indépendant généralement de la nature des objets physiques de la machine sur laquelle le langage est mis en oeuvre. Le langage définit une machine virtuelle composée d'objets informatiques. Pour l'utilisateur, l'usage d'une telle machine offre deux avantages: les objets sont plus variés et on peut en créer un nombre arbitrairement grand. Tout comme les machines physiques, les machines virtuelles sont dotées d'objets à caractère exécutoire que nous appelons procédures primitives.

Représentation d'un objet informatique. L'utilisateur doit représenter les objets sur lesquels il veut effectuer des traitements (par exemple, des objets mathématiques, ou des objets concrets) à l'aide d'objets informatiques. De la même manière, un compilateur du langage représente chaque objet informatique en un objet en mémoire, à l'aide des objets physiques [1 , 2]. On dit que l'objet en mémoire est la représentation de l'objet informatique.

Accès à un objet informatique. Le traitement d'un objet par une procédure primitive est nécessairement précédé d'un accès. Accéder à un objet, c'est d'abord le distinguer parmi d'autres objets; c'est ensuite obtenir l'une de ses caractéristiques. Des tentatives récentes [3] introduisent des objets possédant des caractéristiques dont chacune peut servir soit à les distinguer (valeur d'accès), soit à fournir un élément du traitement (valeur rendue par l'accès). C'est l'approche dite "associative". Cependant la tradition dans le domaine des accès consiste encore à distinguer le concept de valeur d'accès de celui de valeur de traitement. Nous choisissons de rester dans ce cadre. La terminologie que nous adoptons est largement inspirée de la terminologie ALGOL 68 [4 , 5]. Nous avons cependant choisi de modifier certains termes, déjà utilisés avec un sens différent, en essayant de leur substituer un terme qui semble avoir un sens accepté [2 , 6].

Types d'accès. Objets externes - objets internes. L'étude de quelques langages [2 , 4 , 5 , 7 , 8 , 9 , 10] montre différents types d'accès aux objets.

On peut accéder à un objet par l'intermédiaire d'un objet externe. Par opposition l'objet accédé est dit objet interne. Les objets externes fournissent un accès, mais ne peuvent pas en général faire l'objet d'un traitement. Un identificateur, une notation sont des objets externes.

On peut accéder aussi à un objet interne par l'intermédiaire d'un autre objet interne. Un repère (par exemple un nom en ALGOL 68) ou un objet composé (par exemple un tableau en ALGOL 60, une valeur multiple en ALGOL 68) sont des objets internes. On peut accéder à ces objets pour les traiter ou pour les utiliser comme moyen d'accès à un autre objet.

Une procédure est aussi un objet interne. Elle a des propriétés supplémentaires, d'une part celle de pouvoir créer et détruire des objets, d'autre part celle d'effectuer des traitements. Si l'exécution d'une procédure fournit un objet en résultat, on peut la considérer alors comme un moyen d'accès à cet objet.

Caractéristiques des objets internes. Tous les objets internes ont une valeur qui peut faire l'objet d'un traitement. C'est leur caractéristique commune. Un objet interne peut posséder d'autres caractéristiques. Parmi celles-ci,

nous retenons:

- son type: il précise un domaine dans lequel l'objet prend sa valeur;
- sa durée de vie;
- la fonction d'accès à ses constituants s'il s'agit d'un objet composé.

Certaines de ces caractéristiques, dont nous préciserons le sens, peuvent faire l'objet d'un traitement (comparaison du type de deux objets, modification de la fonction d'accès...). Nous ne les assimilons pourtant pas, en général, à des valeurs.

Définitions. De toutes ces observations, nous ne voulons retenir, en première approche, qu'un modèle simple. Dans ce modèle, nous distinguons:

- les objets externes dont le rôle est de *désigner* des objets internes,
- les objets internes ayant des caractéristiques, dont une valeur.

Les seuls objets externes que nous considérons sont les identificateurs. Un identificateur est créé par l'utilisateur ou prédéfini dans le système. Une notation, qui est un moyen universel de *désigner* certains objets internes, primitifs, est considérée comme un cas particulier d'identificateur prédéfini.

Nous considérons ensuite les objets internes, que nous appelons simplement objets. Laissant de côté leur sémantique et leur structuration, nous les classons en quatre grands genres: les objets primitifs et les repères (appelés objets élémentaires), les procédures et les objets composés. Entre ces objets, nous considérons trois types de relations: la relation *repérer* entre un repère et un objet repéré, la relation *fournir* (en ALGOL 68, on dit être un composant de) entre un objet composé et l'un de ses composants, et la relation *délivrer* entre une procédure et son résultat.

On appelle chaîne d'accès à un objet une composition de relations qui *conduit* d'un identificateur à un objet. On dit que l'identificateur I *conduit* à l'objet x s'il existe une relation *désigner* entre I et un objet y tel que:

- soit y est l'objet x lui-même,
- soit il existe entre y et x une composition des relations *repérer*, *fournir* et *délivrer*.

Les mots *désigner*, *repérer*, *fournir*, *délivrer*, *conduire*, seront utilisés dorénavant dans le texte avec le sens qui leur a été donné ci-dessus.

Remarques: 1) Nous considérons une formule comme la représentation d'appels, éventuellement composés, de procédures primitives. Dans la formule $A + B * C$, interprétée comme $+(A, *(B, C))$, $*$ est la notation d'une procédure primitive. $*(B, C)$ *délivre* un objet interne.

2) Nous traiterons, au chapitre 4., du cas des identificateurs, représentés en mémoire sous forme d'une chaîne de caractères, et objets de traitement [8, 10].

2.2. IDENTIFICATEUR ET DESIGNATION.

La relation de désignation a les propriétés suivantes:

- 1) l'association entre un identificateur et l'objet qu'il désigne est effectuée pour une période de temps limitée, généralement bien définie (voir durée de vie, 2.4.). Pendant cette période de temps, l'association est immuable: un identificateur désigne un seul objet.
- 2) un même objet peut être désigné par plusieurs identificateurs (synonymie).
- 3) à un instant donné, un objet peut ne plus être désigné par un identificateur (voir durée de vie, 2.4.). On peut néanmoins accéder à cet objet pourvu qu'il appartienne à une chaîne d'accès construite à partir d'un autre identificateur.
- 4) au cours du temps, un identificateur peut désigner plusieurs objets. Pour un processus donné, cela se traduit par autant d'établissements de relations de désignation, dont une seule est valide pendant une période de temps donnée.

Ainsi, pendant une période de temps donnée et connue, il existe une fonction entre un ensemble d'identificateurs et un ensemble d'objets. C'est le cas par exemple pour les identificateurs définis dans un programme FORTRAN (la période de temps est la durée d'exécution du programme) ou pour les identificateurs définis dans un bloc en ALGOL 60 (La période de temps est celle d'une exécution de bloc).

2.3. OBJETS (INTERNES) ET RELATIONS ENTRE CES OBJETS.

2.31. Objets élémentaires.

Les objets élémentaires sont soit des objets primitifs, soit des repères.

Les objets primitifs: ils sont construits sur les valeurs universelles, de type primitif, définies par le langage. Ils ont un type et une valeur. Leur durée de vie est illimitée.

Exemple: entier, booléen, réel, caractère,...

Les repères: ce sont des objets dont la valeur a pour rôle de repérer un objet quelconque choisi dans un ensemble d'objets du même type, soit μ . Ces objets ont une valeur, un type (appelé "repère de μ ") et une durée de vie limitée.

La relation repérer permet d'accéder à un objet à partir d'un autre objet. La relation établie peut varier au cours du temps: une valeur de type repère de μ peut repérer successivement n'importe quel objet de type μ .

On considère ici qu'un processus peut créer un nombre arbitrairement grand de repères.

Exemples: 1) En ALGOL 60, par la déclaration réel A , on crée un objet de type repère de réel. A désigne cet objet, soit rep_A . Par l'instruction $A := 3.4.$, on fait repérer à rep_A l'entier désigné par 3.4. Une chaîne d'accès à la valeur 3.4. est la suivante:

$$A \xrightarrow{\text{désigne}} rep_A \xrightarrow{\text{repère}} \text{valeur de } 3.4.$$

2) En ALGOL 68, un identificateur peut désigner un objet de type ent, rep ent, rep rep ent, etc..., ce qui signifie qu'il désigne respectivement un entier, un repère d'entier, ou un repère de repère d'entier. Dans le troisième cas, une chaîne d'accès à une valeur entière est la suivante:

$$\text{identificateur} \xrightarrow{\text{désigne}} repère \text{ de } repère \xrightarrow{\text{repère}} repère \xrightarrow{\text{repère}} \text{valeur entière}$$

2.32. Les procédures.

Une procédure est un objet interne ayant une valeur et dont une caractéristique essentielle et supplémentaire est d'être exécutoire.

La valeur d'une procédure est précisée par une notation qui est celle, donnée, d'une procédure primitive, ou qui est construite par composition des constructions permises d'un langage (formules, instructions, blocs,...).

Exemple: $+$ est une procédure primitive d'ALGOL 60, LW est une procédure primitive de langages d'assemblage.

On peut toujours désigner une procédure grâce à un identificateur ou sa notation, on peut parfois la faire repérer ou en faire un composant d'un ensemble. Lorsqu'une chaîne d'accès conduit à une procédure, c'est parfois pour considérer sa valeur au même titre que les valeurs d'un autre type, c'est le plus souvent pour en commander l'exécution.

Au caractère exécutoire d'une procédure sont liées plusieurs notions: celle de création, de manipulation et de destruction d'objets, celle d'établissement de relations avec les objets du monde extérieur, et celle de délivrance d'une valeur. Nous étudions ces caractéristiques en 2.5.

2.33. Objets composés.

Nous n'avons considéré jusqu'ici que des objets et des fonctions d'accès élémentaires. Dans un système, on peut aussi créer des objets composés d'objets élémentaires ou composés, et accéder à chacun de leurs éléments grâce à une fonction d'accès.

Comme notre propos est principalement d'exprimer des opérations macroscopiques effectuées sur des objets ou entre des objets (création, liaison, compilation, destruction), nous évitons de fixer un cadre de définition très précis et trop rigide pour ces objets, comme c'est le cas en ALGOL 68. Au contraire, nous en donnons quelques propriétés générales souhaitables (le chapitre 8 peut être

considéré comme un essai d'expression de ces propriétés dans le cadre du langage ALGOL 68).

Considérons l'ensemble formé d'un regroupement d'objets. Sur cet ensemble, on définit un nouveau type d'objet, appelé objet composé. Dans ce but, on considère une classe d'objets que nous appelons arguments. Soit $\mathcal{A}$ un ensemble d'arguments, $\mathcal{C}$ un ensemble d'objets, et Φ une fonction biunivoque de $\mathcal{A}$ dans $\mathcal{C}$. $\mathcal{A}$, Φ et $\mathcal{C}$ définissent l'objet composé. Soit x cet objet composé, et X un identificateur conduisant à cet objet.

L'ensemble $\mathcal{A}$ peut être un ensemble d'identificateurs ou de valeurs. Si $\mathcal{A}$ est un ensemble d'identificateurs, l'accès à un composant est analogue à une désignation; il s'effectue à partir du couple $(X, \Phi(I))$, $I \in \mathcal{A}$.

Exemple: Une structure de PL/1 est un ensemble de repères. Chaque repère est désigné par un identificateur de champ. Si A désigne la structure, et B l'un de ses champs, $A.B$ fournit un repère. En ALGOL 68, B de A est une écriture équivalente.

Si $\mathcal{A}$ est un ensemble de valeurs, l'accès à un composant est analogue en général à une délivrance de valeur par une procédure; il s'effectue par exemple à partir d'un couple $(X, \Phi(I))$, où I est un identificateur conduisant à une valeur $v \in \mathcal{A}$.

Exemple: Un tableau, en ALGOL 60, est un ensemble de repères. Si T désigne le tableau, $T[I]$ fournit un repère dans l'ensemble. Pour un tableau à deux dimensions, $\mathcal{A}$ est un ensemble de couples d'entiers, etc...

Dans un système, on rencontre fréquemment des objets dits de "taille variable", par exemple un fichier, un tableau ALGOL 68 à bornes flexibles. Nous disons alors que dans le triplet $(\mathcal{A}, \Phi, \mathcal{C})$, la fonction d'accès peut être étendue ou restreinte à un sur-ensemble ou à un sous-ensemble de $\mathcal{A}$, le nombre d'objets de $\mathcal{C}$ variant en conséquence.

On rencontre aussi dans un système des objets composés dont la fonction d'accès est implicite, déduite de l'organisation des objets. Chaque accès s'effectue alors à l'aide, par exemple, de l'identificateur qui désigne l'objet.

Exemple: La fonction d'accès aux composants d'une collection de données peut être implicite si l'organisation du fichier est du type séquentiel. Dans le cas contraire, elle est précisée à l'aide d'index ou de clés. Elle fournit une valeur ou un repère dans la collection.

On rencontre enfin dans un système des objets composés auxquels n'est pas attachée une seule fonction d'accès. Ceci peut signifier que l'objet caractérisé un moment donné par le triplet $(\mathcal{A}, \Phi, \mathcal{C})$ devient l'instant d'après le triplet $(\mathcal{A}', \Phi', \mathcal{C})$. Ceci peut signifier qu'il existe deux objets composés, caractérisés par le même ensemble $\mathcal{C}$: on dit alors que les deux objets partagent le même ensemble d'objets.

Considérons la modification $(\mathcal{A}, \Phi, \mathcal{C}) \rightarrow (\mathcal{A}', \Phi', \mathcal{C})$. Une interprétation consiste à considérer que tout regroupement d'objets a une structure élémentaire $(\mathcal{A}_0, \Phi_0, \mathcal{C})$. Par exemple, $\mathcal{A}_0$ est l'ensemble des entiers $\{1, 2, \dots, \text{card}(\mathcal{C})\}$. L'objet composé tel que nous l'avons défini est alors un couple d'objets internes: le premier est une procédure, ou un objet analogue (par exemple format); le second est le regroupement $(\mathcal{A}_0, \Phi_0, \mathcal{C})$ d'objets ordonnés. Si l'on veut exprimer la modification souhaitée, il suffit de remplacer le premier champ du couple par un objet de type repère de procédure (ou repère de format).

Exemples: 1) Les fichiers ALGOL 68 sont une valeur structurée composée, entre autres, d'un champ de type repère de format, et d'un champ donnant accès à un tableau de caractères.

2) Les fichiers dans SAM [16] sont formés d'une collection ordonnée d'objets et d'une procédure d'accès interchangeable.

Une interprétation plus restrictive consiste à définir un objet composé comme le $(n+1)$ - uplet $((\mathcal{A}_1, \Phi_1), (\mathcal{A}_2, \Phi_2), \dots, (\mathcal{A}_n, \Phi_n), \mathcal{C})$. Au moment où l'objet est rendu accessible à un processus (nous appellerons cette opération

une liaison), on précise le couple $(\mathcal{A}_i, \Phi_i)$ choisi pour accéder à l'objet.

Exemple: Dans certains systèmes de fichiers, on peut préciser, à l'ouverture d'un fichier (c'est l'opération de liaison) qu'on choisit d'accéder à ses composants de manière séquentielle ou de manière indexée.

Nous considérons que cette deuxième interprétation est meilleure, car elle limite strictement la classe des fonctions d'accès définissables sur l'ensemble d'objets. Ceci est important d'un point de vue protection: en effet, au moment de la liaison de l'objet, un mécanisme de protection peut interdire l'usage de telle fonction d'accès.

2.4. DUREE DE VIE D'UN OBJET ET D'UNE CHAÎNE D'ACCES.

Rappelons que l'utilisateur ne connaît que des identificateurs (au sens large) pour exprimer ses algorithmes. La propriété essentielle d'un identificateur est qu'il permet d'accéder d'une part à l'objet qu'il désigne, et d'autre part à tout objet d'une chaîne d'accès construite à partir de lui. C'est généralement la position syntaxique qui précise l'objet accédé. En ce sens, on peut dire que l'identificateur est la représentation externe d'un objet quelconque d'une chaîne d'accès.

Exemple: Considérons, en ALGOL 60, un objet rep_I de type repère d'entier, désigné par I .

L'instruction

$$I := \langle expression \rangle$$

fait repérer à rep_I la valeur entière fournie par l'élaboration de $\langle expression \rangle$.

Soit J désignant un second objet, rep_J , de type repère d'entier.

L'instruction

$$J := I$$

fait repérer à rep_J non pas l'objet désigné par I , mais l'objet repéré par rep_I .

Dans

$$I \xrightarrow{\text{désigne}} rep_I \xrightarrow{\text{repère}} i$$

I conduit à rep_I et à i .

Un objet de type primitif a une valeur dont il existe une notation universelle: on peut toujours mettre en relation l'un de ses exemplaires. Par contre, un repère est généralement créé pour une durée de vie limitée et détruit par une opération explicite ou implicite: nous dirons alors qu'il est mort.

La relation de désignation entre un identificateur et un objet est établie pour une durée limitée, parfois indépendante de la durée de vie de l'objet. La relation de repérage entre deux objets est, elle aussi, momentanée. Lorsqu'il n'existe plus de chaîne d'accès à un objet, nous disons qu'il est inaccessible.

Pour des raisons de fiabilité, la durée de vie d'un objet doit être supérieure ou égale à l'existence d'une chaîne d'accès conduisant à lui.

2.5. PROCEDURE ET CONTEXTE.

Nous examinons maintenant les propriétés exécutoires d'un objet-procédure [12]. Il est convenu d'appeler processus l'exécution d'un objet-procédure particulier: le programme d'un utilisateur, ou une procédure s'exécutant en parallèle avec d'autres procédures. Pourtant, à proprement parler, un processus se décompose en processus élémentaires, chaque processus élémentaire étant l'exécution d'une procédure primitive. Aussi, considérant un objet-procédure $\mathcal{P}$, nous appelons processus ($\mathcal{P}$) son exécution.

2.51. Objets manipulés par un processus.

Un processus peut créer des objets: repères, objets composés, procédures. Il établit des relations de désignation entre des identificateurs locaux et les objets qu'il crée.

Un processus peut en outre utiliser des objets de type quelconque, qui ont une existence indépendante de son exécution. Chacun de ces objets appartient, par définition, à une chaîne d'accès construite à partir d'un identificateur. Pour accéder à l'objet, le processus utilise cet identificateur: on dit alors qu'il

est global à la procédure, ou utilise un identificateur de paramètre formel qu'il met en relation avec l'objet. Tout objet de type primitif appartient à la première catégorie.

Un processus peut enfin détruire les objets qu'il a créés. Il peut aussi en rendre d'autres inaccessibles.

2.52. Lexique, contexte, état d'exécution.

Dans la plupart des systèmes, on définit certains objets pour une durée limitée seulement par la durée de vie du système. Ces objets (repères, objets composés, procédures) sont désignés par des identificateurs, utilisables par tous les processus du système. Nous assimilons dorénavant ces objets aux objets de type primitif et leurs identificateurs à des notations d'objets primitifs.

Exemples: 1) L'ensemble des instructions d'une machine, ses registres généraux et les objets du moniteur forment avec les constantes de divers types les objets primitifs d'un système basé sur un langage d'assemblage.

2) L'ensemble des procédures standards et certains objets définis pour les entrées-sorties forment, avec les constantes de type réel, entier, booléen, les objets primitifs d'un système basé sur ALGOL 60.

On appelle lexique un ensemble d'identificateurs, et on définit le lexique primitif du système comme l'ensemble des identificateurs conduisant à des objets primitifs.

Considérons l'ensemble $\mathcal{D}(T, \mathcal{L})$ des objets désignés, pendant une période de temps T , par les identificateurs d'un lexique $\mathcal{L}$. On appelle contexte le couple $(\mathcal{L}, \mathcal{D}(T, \mathcal{L}))$ dans lequel chaque objet de $\mathcal{D}(T, \mathcal{L})$ est en correspondance avec l'identificateur de $\mathcal{L}$ qui le désigne. Le contexte primitif est construit une fois pour toutes à partir du lexique primitif et est constant pendant toute la durée de vie du système.

Enfin, considérons l'ensemble $\mathcal{X}(t, \mathcal{D}(T, \mathcal{L}))$ des objets appartenant à l'instant $t \in T$, à une chaîne d'accès construite à partir d'un identificateur de $\mathcal{L}$. On dit que x est l'état d'exécution associé au contexte construit sur $\mathcal{L}$. L'état d'exécution associé au contexte primitif varie pendant la durée de vie du système.

2.53. Lexiques, contextes et états d'exécution associés à une procédure.

Lorsqu'une procédure $\mathcal{P}$ est appelée par un processus $\mathcal{Q}$, elle trouve un certain état d'exécution que son exécution peut modifier. Celui-ci est construit sur un contexte qui est l'union des trois contextes suivants.

a) Le contexte primitif.

b) Un contexte particulier, propre à $\mathcal{P}$ ou qu'elle partage avec d'autres procédures. Le lexique à partir duquel ce contexte est construit contient l'ensemble des identificateurs globaux de la procédure $\mathcal{P}$.

c) Un contexte construit par le processus qui appelle $\mathcal{P}$ à partir du lexique des paramètres formels de $\mathcal{P}$. Nous appelons ce contexte le contexte des paramètres.

Pour qu'une procédure puisse être exécutée, il est nécessaire que les trois contextes cités soient construits.

Lorsque le contexte particulier d'une procédure $\mathcal{P}$ a été construit, nous disons que la valeur de la procédure peut être élaborée. Sur une même notation de procédure, on élabore une nouvelle valeur chaque fois que l'on redéfinit son contexte particulier. Une procédure a un caractère universel si son contexte particulier est vide. Sa valeur est alors constante.

Lorsque le contexte des paramètres d'une procédure $\mathcal{P}$ est construit, il peut y avoir création d'un processus ($\mathcal{P}$).

2.6. FICHIERS, PROCESSEURS, MODULES.

Nous avons présenté dans les paragraphes précédents différents types d'objets et leurs caractéristiques. Les objets appelés fichiers, processeurs, modules, que l'on a coutume de rencontrer dans un système sont-ils réductibles à ces objets? Nous ne répondons pas dans ce paragraphe de façon formelle: nous essayons simplement de les caractériser un peu mieux.

2.61. Les fichiers.

Un fichier est un objet composé, conservé dans le système et utilisable par toute une classe de processus. Dans la plupart des systèmes, on définit les fichiers et leur mode d'utilisation à l'aide des concepts suivants (voir [5], [12]).

a) organisation

- organisation des supports de fichier.
- organisation logique du fichier.

b) accès

- caractéristiques d'accès du support.
- fonctions d'accès aux composants du fichier.

c) utilisation

- gestion des fichiers.
- opérations sur les composants.

d) contrôle des incidents et des erreurs.

1°) Dans une première étape, on cherche à libérer l'organisation du fichier, ses caractéristiques d'accès et les opérations possibles sur ses composants de la structure technologique de leurs supports. Ce degré de liberté est d'autant plus souhaitable que la collection d'objets composant le fichier est de plus en plus souvent représentée, au moment de l'accès, sur le même support de mémoire que les autres objets manipulés par un processus. A ce niveau, le fichier se caractérise par:

- une organisation logique: elle définit un ensemble d'objets élémentaires et les caractéristiques de chacun d'eux;

- une ou plusieurs fonctions d'accès aux composants du fichier: c'est le mode d'adressage;
- les opérations possibles sur les composants du fichier: lecture, écriture;
- des opérations de gestion: création, destruction, conservation, ouverture, fermeture.

2°) Dans une seconde étape, on cherche à homogénéiser l'organisation des fichiers avec celle des objets que l'on peut définir dans un langage de programmation, et on souhaite étendre aux composants du fichier les opérations du langage.

Si l'organisation est celle d'un objet composé du langage, et que la fonction d'accès, unique, s'en déduit, on considère alors le fichier comme un objet composé.

Si l'organisation est celle d'un objet composé (un vecteur par exemple), sur lequel sont définies plusieurs fonctions d'accès (accès par adressage et accès séquentiel par exemple), nous pouvons considérer deux cas.

- a) Si toute utilisation de l'objet par un processus donné passe par une seule fonction d'accès, exprimable dans le langage, on cherche à se ramener au cas précédent.
- b) Dans le cas contraire, on considère le fichier comme l'ensemble formé:
 - de la collection de données exprimée comme un objet composé $\mathcal{X}$: c'est au moins une suite ordonnée de caractères;
 - de procédures dont les contextes particuliers se partagent $\mathcal{X}$ (voir 2.33.).

L'accès au fichier passe alors nécessairement par l'une des procédures.

3°) Enfin, dans une dernière étape, on cherche aussi à exprimer les opérations de gestion de fichiers à l'aide des outils définis par le langage. Par exemple, l'ouverture d'un fichier est assimilable à l'élaboration de la valeur d'un objet suivie de sa mise en relation.

2.62. Les processeurs.

Un processeur est généralement défini comme un mécanisme câblé ou programmé, capable de mener à bien des processus, c'est-à-dire des exécutions de procédures.

De notre point de vue, et conformément aux définitions données en 2.5., un processeur est un état d'exécution, construit sur un contexte, et contenant tous les objets rendant possible l'exécution d'une classe de procédures.

On appelle processeur primitif un processeur dont le contexte ne contient que des objets primitifs. Un processeur câblé est primitif; le contexte associé est formé d'instructions désignées par un numéro, et d'emplacements (ou registres) désignés par une adresse. L'état d'exécution qui définit ce processeur résulte de l'état défini par l'opération dite de restauration.

Un processeur ALGOL 60 est défini par l'état d'exécution construit à partir de ses opérateurs et procédures standards. On peut le représenter comme un bloc englobant tout programme ALGOL 60. En ALGOL 68, un tel bloc est appelé prologue [4, 5, 12]. Un processeur ALGOL 68 résulte de l'exécution d'un prologue. Un tel processeur est appelé primitif s'il définit à lui seul la machine virtuelle d'un utilisateur.

Lorsqu'un utilisateur a écrit, en ALGOL 60,

début

procédure P ; début <corps de P> fin ;

entier x ;

il a étendu l'état d'exécution dans lequel les objets qu'il va définir vont être traités. Son processeur initial, le processeur ALGOL 60, s'est enrichi.

Ainsi, très informellement, on peut considérer un processeur comme une procédure incomplète représentée par une chaîne de caractères *s* telle que

s s' fin de la procédure

où *s'* est une chaîne représentant un ensemble éventuellement vide d'instruction,

soit une procédure. *s'* représente un traitement effectué par le processeur *s*.

On peut vouloir évidemment restreindre s' à ne désigner qu'un sous-ensemble des objets du contexte défini par s . (Exemple: un processeur câblé, un processeur d'édition de texte, ...). Ce sous-ensemble est le contexte visible du processeur.

Nous retenons donc provisoirement qu'un processeur est un objet dont la valeur s'assimile à une exécution incomplète d'une procédure.

2.63. Les modules.

Parnas [14] définit un module comme un objet composé

- " - de variables d'état , consultables mais non modifiables,
- de procédures d'utilisation,
- et de procédures et variables internes, inaccessibles de l'extérieur du module, mais manipulables par les procédures d'utilisation."

Conformément à notre terminologie, un module peut être considéré comme un objet composé

- de procédures d'accès, qui délivrent des valeurs (consultation des variables d'état) ou qui effectuent des actions (utilisation du module)
- et d'un ensemble quelconque d'objets.

Ainsi, un module est un cas particulier de processeur dont le contexte visible ne contient que des procédures.

Un fichier, considéré comme un ensemble formé

- d'une collection de données,
- et de procédures d'accès,

est analogue à un module, à une différence près toutefois. Cette différence concerne la durée de vie de la collection de données, qui doit être identique ou supérieure à celle de l'objet-module.

Nous voyons que tous ces objets, présentés informellement, ont une certaine ressemblance. Il nous reste à exprimer, à l'aide d'objets ayant une définition précise, des caractéristiques et des propriétés semblables. C'est ce que nous avons fait dans le cadre du langage ALGOL 68 [15] et que nous présentons au chapitre 8.

BIBLIOGRAPHIE

- [1] PAIR C.
Structure de Données
Ecole d'été AFCET (1971)
- [2] PAIR C.
Compilation
Ecole d'été AFCET (1972)
- [3] ABRIAL J.R.
Structure de Données et de Programmes
Cours. Université de Grenoble (1972)
- [4] VAN WIJNGAARDEN, MAILLOUX B.J., PECK J.E.L., KOSTER C.H.A.
Report on The Algorithmic Language ALGOL 68
Springer Verlag - Berlin (1969)
- [5] BACCHUS P., PAIR C., PECCOUD D. (Editeurs)
Manuel ALGOL 68 , par le groupe Algol de l'AFCET
A paraître. HERMANN
- [6] WEGNER P.
Data Structure Models for Programming Languages
in Proc. of a Symposium on Data Structures in Programming Languages
SIGPLAN NOTICES (1971)
- [7] GRISWOLD R.
The SNOBOL 4 Programming Language
Prentice Hall (1968)
- [8] LUCAS P., WALK K.
On The Formal Description of PL/1
Annual Review of Automatic Programming, 6, 3 (1969)

- [9] Mc CARTHY J. et al.
The LISP 1.5 Programmers Manual
M.I.T. Press - Cambridge (1965)
- [10] NAUR P. et al.
Revised Report on the Algorithmic Language ALGOL 60
CACM 6,1 (1963)
- [11] CROCUS (Ed.)
Les Systèmes d'Exploitation des Calculateurs - Principes de Conception
par un groupe de rédaction
A paraître . DUNOD
- [12] VERJUS J.P.
Quelques Propriétés des Langages d'Utilisation de Systèmes
Ecole d'été AFCET (1973)
- [13] HORNING J.J., RANDELL B.
Process Structuring
Comp. Survey 5,1 (1973)
- [14] PARNAS D.L.
A Technique for Software Module Specifications with examples
CACM 15,5 (1972)
- [15] TRILLING L., VERJUS J.P.
Introduction Dynamique de Processeurs dans un Environnement Conversationnel
Article proposé pour Publication (1973) - (pièce jointe)
- [16] ROSSIENSKY J., TIXIER V.
SAM : The File System
Document interne - ENSA

CHAPITRE 3. CONTRAINTES APPORTEES PAR LE SYSTEME.

3.1. INTRODUCTION

Lorsque l'utilisateur écrit un algorithme, il utilise des identificateurs (au sens large). Le langage utilisé doit lui permettre de se dégager le plus possible des contraintes de mise en oeuvre sur un calculateur. On peut classer ces contraintes sous plusieurs rubriques.

- représentation des objets et des relations en mémoire.

L'utilisateur n'est contraint que par des limitations technologiques qui fixent par exemple une limite au domaine de représentation des valeurs primitives, au nombre de repères qu'il peut créer. Ces contraintes étant données, il faut que la mise en oeuvre traduise en une représentation en mémoire les objets et relations définis par le langage.

- conservation et partage des objets.

Dans un programme, un objet est créé et mis en relation avec d'autres objets. Si le système autorise la conservation et la réutilisation d'un objet, il peut être nécessaire de considérer le contexte dans lequel il a été défini. En outre, l'utilisateur est soumis aux contraintes d'utilisation de certains objets de la communauté dont l'usage n'est pas totalement libre. Cependant, c'est à son insu que le système réalise un partage réel de certains objets, c'est-à-dire qu'il en autorise l'utilisation simultanée par plusieurs processus. Cette contrainte est due à la limitation de la taille des mémoires utilisées comme support et à la plus grande fiabilité qui résulte de la gestion en un seul exemplaire de ces objets.

- diversité des supports.

Cette contrainte doit être ignorée de l'utilisateur. En particulier, il ne s'intéresse qu'à la structure logique des objets qu'il définit. Une procédure spéciale du système - appelons-la compilateur - convertit les objets, que l'utilisateur définit sur un support externe grâce à des identificateurs, en une représentation interne (voir chap. 4.).

3.2. REPRESENTATION DES OBJETS EN MEMOIRE.

La mémoire d'un calculateur est constituée d'emplacements qui ont, à tout instant, un contenu. Les emplacements sont les objets internes du processeur câblé qui les manipule. Il désigne ces emplacements grâce à un nom, et nous verrons que la relation de désignation entre un nom et un emplacement a les mêmes propriétés que la relation de désignation entre un identificateur et un objet d'un langage. Un nom permet au processeur d'accéder au contenu de l'emplacement, qui peut être interprété comme un exemplaire d'une valeur d'un certain type: instruction, valeur entière, ... ou à nouveau comme un nom. Le processeur peut modifier le contenu de certains emplacements. Il y a donc une relation modifiable entre un emplacement et un ensemble de contenus: c'est la relation contenir.

Nous considérons ici qu'un emplacement a une taille fixe.

Dans cet ouvrage, nous considérons que le langage d'utilisation d'un système n'est pas en général celui de la machine. Aussi, les emplacements de mémoire ne sont pas considérés comme des objets internes, mais comme des entités inconnues de l'utilisateur, utilisées pour représenter des objets internes. Il est nécessaire de représenter d'une part les objets, c'est-à-dire leur valeur et éventuellement certaines de leurs caractéristiques, et d'autre part les relations entre ces objets. La représentation d'un identificateur et celle de la relation de désignation sont étudiées au chapitre 4.

Les objets primitifs comme les entiers ne sont pas conservés en mémoire: on sait en construire un exemplaire au fur et à mesure des besoins. Ainsi, d'une façon générale, les emplacements et contenus vont être utilisés, à la demande, pour représenter un objet ou une relation. Lorsqu'un objet devient mort ou inaccessible (2.4.), les emplacements désormais inutiles sont récupérés pour d'autres usages. Nous supposons qu'une opération appropriée sait trouver un emplacement à la demande.

3.21. Représentation de la relation repérer.

La relation repérer se représente par la relation contenir qui existe entre un emplacement et son contenu à un instant donné. La représentation d'un objet de type repère utilise donc un emplacement de taille suffisante pour contenir la représentation de tout objet qu'il peut repérer. Le nom qui désigne un tel emplacement représente la valeur du repère. Tout exemplaire de cette valeur peut à son tour être contenu dans un emplacement.

3.22. Notion de descripteur et de préfixe.

Considérons un objet dont la représentation occupe les contenus de n emplacements. Si cet objet est repéré, on peut imaginer que la représentation du repère doive utiliser n emplacements. Pour faciliter l'établissement des relations de repérage (il en est de même pour les autres relations), on cherche à définir une information de taille fixe et réduite pour décrire l'objet. En outre, la multiplicité des exemplaires d'une même valeur conduit elle aussi à définir une information de taille réduite pour décrire l'objet.

On appelle descripteur cette information. Le composant essentiel du descripteur d'un objet est une information qui permet de mettre en oeuvre un chaînage avec l'ensemble d'emplacements utilisé pour représenter l'objet. Un chaînage est une fonction d'accès en mémoire: il prend généralement la forme d'un couple (nom de l'emplacement origine de l'ensemble, taille de l'ensemble). Le nom utilisé pour mettre en oeuvre un chaînage ne doit pas être assimilé à la représentation d'une valeur de type repère.

Le descripteur, d'une façon générale, est composé des informations précisant les caractéristiques de l'objet autres que sa valeur: type, fonction d'accès, durée de vie, caractéristiques exécutoires. Certaines caractéristiques de l'objet sont représentées parfois, non dans son descripteur, mais dans l'ensemble d'emplacements. Une raison de ce choix est le souci de pouvoir représenter un descripteur par une information de taille fixe, généralement égale à la taille d'un emplacement.

Si l'ensemble des caractéristiques (y compris la valeur) sont représentables sur un seul emplacement, le chaînage est inutile. On appelle alors préfixe l'ensemble des caractéristiques de l'objet, autre que sa valeur. Le couple (préfixe, valeur) est représentable dans un seul emplacement. Le préfixe peut évidemment ne pas exister.

Nous verrons aux chapitres 5 et 6 des illustrations de représentation d'objet.

3.23. Eléments d'un descripteur ou d'un préfixe.

Afin de préciser les éléments qui entrent en jeu dans la représentation des objets, examinons leurs caractéristiques, autres que la valeur.

1°) Le type.

Le type précise la nature de l'objet représenté et les opérations dont il peut être un opérande. Il peut être testé à l'occasion de certains accès (affectation, opération, passage d'arguments), notamment lorsque la validité de ces accès n'a pas pu être vérifiée à la compilation des procédures qui utilisent l'objet correspondant.

2°) La durée de vie.

La durée de vie d'un objet peut être parfois déduite de l'organisation des zones où sont réservés les emplacements utilisés. Elle peut être aussi spécifiée explicitement par une instruction. Dans ce cas, il est nécessaire de représenter l'indication de durée de vie. Celle-ci est alors utilisée afin, d'une part, d'effectuer des contrôles sur la validité de certaines opérations et, d'autre part, d'effectuer au moment opportun la récupération des emplacements morts ou inaccessibles.

3°) La fonction d'accès.

La fonction d'accès est l'ensemble des informations nécessaires pour atteindre chaque élément d'un objet composé: par exemple, l'origine d'un tableau et les valeurs de bornes de chacune de ses dimensions.

Pour représenter une fonction d'accès, on cherche à la décomposer en fonctions d'accès câblées en mémoire, l'indexation et le chaînage par exemple. Si ce

n'est pas possible, on la représente sous la forme d'une tabulation. C'est la fonction d'accès qui détermine souvent le choix d'un mode de représentation d'un objet composé (voir les illustrations données en 5.54. et 6.32.).

Remarque: Nous abordons au chapitre 4. la représentation des objets-procédés.

3.3. CONSERVATION ET PARTAGE DES OBJETS.

3.31. Désignation des emplacements en mémoire.

3.311. La Mémoire fictive.

Les mémoires physiques utilisées par les systèmes sont limitées en taille et ont des caractéristiques d'accès très différentes. En particulier, l'existence de supports secondaires sur lesquels les processeurs centraux ne peuvent exécuter des instructions impose la mobilité des objets dans les mémoires. Cette mobilité complique les problèmes d'accès aux objets et leur gestion en y mêlant la gestion des supports physiques. On observe, en particulier, le problème suivant. Lorsqu'un objet a été créé, on l'a représenté à l'aide, par exemple, d'un emplacement; un nom désigne cet emplacement, et cette relation doit rester immuable. Si l'on doit utiliser un autre emplacement, il faut alors modifier tous les exemplaires du premier nom. C'est pourquoi nous considérons ici qu'un emplacement de mémoire n'est pas lié à une position de mémoire désignée par une adresse, mais est mobile sur une hiérarchie de mémoires. Un nom est une information permettant de désigner l'emplacement quelle que soit la position de mémoire qu'il occupe. C'est la tâche de l'allocationneur de mémoire de le convertir en une adresse. Nous appelons un tel ensemble d'emplacements une mémoire fictive. Une adresse fictive est un nom d'emplacement.

Ainsi, à un certain niveau d'observation, un processus fait abstraction de la diversité des supports physiques utilisés dans le système.

3.312. La Mémoire segmentée.

Cette simplification de la méthode de désignation des emplacements n'est pas suffisante. Considérons par exemple un objet composé, représenté sur un ensemble d'emplacements contigus. Les noms de ces emplacements sont des numéros séquentiels. Si la taille de l'ensemble croît, il est souhaitable de conserver le caractère contigu des emplacements: on dit qu'ils forment un segment. Par contre, il n'est pas nécessaire de conserver aux emplacements environnant ce segment la propriété de contiguïté avec lui. On définit ainsi un segment comme une suite d'emplacements numérotés 0, 1, 2, ...: le nom d'un emplacement d'un segment est son numéro (on dit aussi indice dans le segment). Entre deux indices dans deux segments, il n'y a aucune relation d'ordre.

Ainsi, à un certain niveau d'observation, la mémoire est constituée de segments. Un segment sert à représenter un objet composé; c'est dans un segment que l'on regroupe des objets de même nature, de même durée de vie, sur lesquels on veut appliquer les mêmes restrictions d'accès, etc.

Un segment est décrit par son descripteur qui est composé de certaines caractéristiques du segment et d'un chaînage sur ce segment.

3.313. Les Descriptifs. Adresse segmentée.

Le nom de tout emplacement d'une mémoire segmentée a pour composante son indice dans un segment. Le couple (descripteur, indice) est un nom d'emplacement si les informations constituant le descripteur sont constantes. Or le descripteur peut ne pas avoir cette propriété.

Si le concept de mémoire fictive n'est pas mis en oeuvre, le chaînage utilisé dans le descripteur est une adresse en mémoire physique, qu'il est nécessaire de mettre à jour à chaque mouvement du segment.

Si les caractéristiques du segment varient au cours du temps, par exemple sa taille, on est confronté au même problème.

Considérons un segment particulier appelé descriptif, groupant des descripteurs de segment. Chaque descripteur a un emplacement attribué dans ce descriptif pendant toute la durée d'utilisation du segment par un processus. Le couple

(*<indice dans le descriptif d'un descripteur de **S**>* ,
*<indice d'un emplacement dans **S**>*)

est constant pendant la période d'utilisation de **S** par un processus. C'est un nom, appelé généralement adresse segmentée.

3.32. Partage des objets.

L'ensemble des segments constitue l'ensemble des objets internes connus du système. Soit $\mathcal{P}$ la procédure qu'un utilisateur veut exécuter pendant une session. Cette activité se traduit par un processus ($\mathcal{P}$). On définit l'espace adressable d'un processus comme l'ensemble des emplacements utilisés pour représenter les objets de son état d'exécution. C'est un espace de taille variable au cours du temps, mis en oeuvre sur un ensemble de segments.

Parmi les segments connus du système, certains sont accessibles à plusieurs processus. Nous disons que ces segments sont partagés. Le partage d'un segment signifie ici que la totalité du segment appartient à l'espace adressable des processus qui y accèdent. Etant données les limitations de taille des mémoires utilisées et compte tenu des difficultés entraînées par la mise à jour de plusieurs exemplaires d'un même segment, on impose comme contrainte aux segments d'être en un seul exemplaire.

Le partage d'un segment résulte du partage d'un objet qui est et doit rester transparent à l'utilisateur (partage d'un compilateur, d'une procédure standard) ou qui, au contraire, répond à une nécessité logique (utilisation d'un fichier comme une boîte aux lettres).

Un segment a des caractéristiques, représentées dans son descripteur. Ces caractéristiques se déduisent de celles de l'objet (ou des objets) qu'il permet de représenter: le type, la taille du segment, sa durée de vie. Un segment a aussi un contenu, représentation d'objets internes. Toutes ces informations ne sont pas nécessairement partageables.

La taille du segment est unique. C'est une information partageable. Par contre, la fonction d'accès aux constituants d'un objet composé (y compris les restrictions d'accès) n'étant pas nécessairement unique, il doit éventuellement en exister une représentation pour chaque processus.

Le segment lui-même doit être rendu partageable. C'est un choix influant sur le mode de représentation des objets que nous étudions au chapitre 4. Pour introduire ce problème, considérons un emplacement du segment, et supposons qu'il doive contenir, lors de son utilisation par un processus donné, la désignation d'un objet propre au processus. Si cette désignation est représentée par un nom d'emplacement, elle n'est pas partageable. On dit que sa valeur est liée à l'espace adressable du processus. Dans le cas contraire, on dit qu'elle est libre.

On résume ces observations en définissant deux états d'un segment partageable, conservé dans le système:

- à l'état libre, le segment possède des caractéristiques intrinsèques,
- à l'état lié (sous-entendu à un processus) le segment possède en outre des caractéristiques propres au processus.

Considérons les valeurs contenues dans le segment. Dans certains systèmes, la liaison du segment à un processus entraîne la création d'un exemplaire dans lequel on substitue à toute valeur libre une valeur liée. L'exemplaire libre du segment est conservé en même temps que les exemplaires liés. Nous avons précisé que nous rejetons cette méthode. Nous verrons qu'il est possible de ne conserver qu'un exemplaire du segment.

Considérons les autres caractéristiques, et supposons qu'elles peuvent être regroupées en un descripteur (autrement dit, on suppose que la taille du descripteur est suffisante pour les renfermer). On peut envisager deux méthodes de partage.

a) Un descripteur libre unique contient toutes les informations intrinsèques, parmi lesquelles l'adresse du segment en mémoire fictive (ou mémoire physique). Ce descripteur est rangé dans le descriptif des objets connus du système et a donc un nom. Dans chaque processus on utilise pour désigner le segment un

nouveau descripteur contenant:

- le nom du descripteur libre,
- les informations propres au processus.

Cette méthode nécessite donc un double chaînage lors de l'accès au segment.

b) Dans chaque processus, on utilise, pour désigner le segment, une copie du descripteur libre, complétée par les informations propres au processus. Cette méthode évite un chaînage, mais soulève quelques difficultés si les caractéristiques du segment varient (par exemple la taille, l'adresse en mémoire physique): en effet il faut alors mettre à jour tous les exemplaires du descripteur, dans tous les descriptifs de processus qui le contiennent.

3.33. Conservation des objets - Catalogue.

Considérons un segment utilisé pour représenter un objet composé partageable. Le processus qui a créé l'objet l'a mis en relation dans une chaîne d'accès pour pouvoir y accéder. Pour que cet objet puisse être utilisé par un autre processus, même si le processus créateur est terminé, il faut satisfaire deux conditions:

- (1) l'objet a une durée de vie indépendante de celle du processus.
- (2) il existe une chaîne d'accès conduisant à cet objet.

Le langage doit donc avoir la propriété d'attribuer une durée de vie potentiellement illimitée à un objet (condition 1).

Pour satisfaire la seconde condition, il suffirait que le langage permette à un processus d'établir une relation de désignation entre un identificateur, utilisable par tous les processus dans le système, et l'objet. Ceci signifierait que le contexte primitif est une entité dynamique conservée dans le système. Cette solution n'est pas retenue, car elle attribue le même pouvoir d'utilisation de l'objet à tous les processus. Pour contrôler l'utilisation de l'objet, on le place sous le contrôle d'un "bibliothécaire". Celui-ci est représenté par une fonction d'accès à un objet composé de tous les objets conservés dans le système. Cet objet composé est appelé catalogue.

BIBLIOGRAPHIE.

[1] VERJUS J.P.

Quelques Propriétés des Langages d'Utilisation de Systèmes
Ecole d'été AFCET (1973)

[2] TRILLING L., VERJUS J.P.

Notions sur l'Implémentation d'ALGOL 68
NT 005 - Rennes (1972)

CHAPITRE 4. COMPILATION. EDITION DE LIEN.

4.1. INTRODUCTION.

Etant donné un choix d'une représentation en mémoire des objets et relations définis par un langage, la tâche d'un compilateur est d'effectuer la "partie statique" de l'exécution d'un programme écrit dans ce langage, et de le transformer en un programme objet confié à un processeur adéquat. Par partie statique d'une exécution, on entend

- d'une part tous les contrôles syntaxiques et sémantiques qui peuvent être faits par un simple examen des caractéristiques connues et constantes des objets. On ne s'intéresse pas ici à cet aspect.
- d'autre part, toutes les élaborations de valeurs qui peuvent être connues à ce stade et l'établissement des relations statiques entre objets. Les relations de désignation et certaines fonctions d'accès ont parfois un caractère statique.

Nous nous intéressons dans ce chapitre à l'établissement des relations de désignation et à l'élaboration de valeurs.

4.2. ETABLISSEMENT DE RELATIONS DE DESIGNATION.

Etablir une relation de désignation, c'est mettre en oeuvre le premier maillon d'une chaîne d'accès conduisant d'un identificateur (objet externe) à un objet (interne).

D'une manière générale, une relation de désignation ne peut s'établir qu'au stade de l'exécution, comme le montrent les exemples suivants.

Exemples: 1) Considérons, en ALGOL 68, les déclarations

ent $X = 3 * N$

ou

ent $X = 3 * (\text{lire } (N) ; N)$

Dans ces deux exemples, l'expression qui délivre la valeur désignée n'est calculable qu'à l'exécution.

2) Dans tous les langages où la gestion de l'espace adressable est dynamique (ALGOL 60, PL/1, ALGOL 68,...), on ne peut connaître avant l'exécution le nom de l'emplacement destiné à représenter un repère.

3) Dans tous les langages, un paramètre formel désigne un objet existant, différent à chaque appel de la procédure. Soit P une procédure de paramètre formel F . Soit X un identificateur désignant un objet $\mathfrak{X}$ représenté par un nom x . $P(X)$ fait désigner à F l'objet $\mathfrak{X}$. Ce n'est qu'à l'exécution que s'établit la relation entre F (ou sa représentation) et x , grâce à un chaînage par exemple.

4) Considérons un identificateur de fichier. Nous avons vu en 3.33. que l'identificateur de fichier est traité comme une chaîne de caractères, argument d'une fonction d'accès f à un objet composé de tous les fichiers. On note à ce propos que dans certains systèmes de fichiers, l'identification d'un fichier prend syntaxiquement la forme d'une chaîne de caractères.

La relation

f (<chaîne de caractères>) $\xrightarrow{\text{fournit}}$ objet "fichier"

doit être mise en oeuvre à l'exécution. Une première raison est que le pouvoir d'un processus à utiliser un fichier ne peut être effectué que lorsque l'on connaît l'identité du processus. Celle-ci est généralement celle de l'utilisateur qui a activé le processus. Une seconde raison interdit de résoudre cet accès au stade de la compilation: l'objet fichier peut ne pas encore exister, ou, s'il existe, ne plus exister au moment de l'exécution.

Remarque: Dans les langages où un identificateur est considéré comme un objet interne, la relation de désignation est évidemment mise en oeuvre à l'exécution. En effet, le résultat d'un calcul sur une chaîne de caractères, soit "*them*" + "*ròc*", peut servir à désigner l'objet dont l'identificateur est *THEMROC*.

Ces exemples nous montrent que l'accès aux objets, par l'intermédiaire d'un identificateur, nécessite en théorie la conservation, au stade de l'exécution, des contextes. Dans ces conditions, on considère le contexte comme un objet composé (comme le catalogue; voir 3.33.) et la relation de désignation est une fonction d'accès dans un contexte.

Soit s la chaîne de caractères représentant en mémoire un identificateur d'un contexte C . Une fonction d'accès $f(s)$, définie sur le lexique de C , fournit l'objet désigné.

C'est cette technique qui est réalisée dans les méthodes d'exécution appelées interprétation. C'est une technique coûteuse, car chaque accès nécessite une recherche dans une table.

A l'opposé de cette technique, il y a celle qui est utilisée dans la compilation des langages d'assemblage. Pour ceux-ci l'objet désigné par un identificateur est connu au stade de la compilation: c'est une valeur dite immédiate ou le nom d'un emplacement. Le contexte, utilisé au stade de la compilation, devient inutile à l'exécution puisque, à tout identificateur apparaissant en paramètre d'une instruction, on a substitué la représentation de l'objet désigné. On dit que l'objet a été lié. Cette seconde méthode est non seulement inapplicable, dans de nombreuses situations, mais en outre on rend difficile la réutilisation des objets (procédures par exemple), qui utilisent un tel mode de désignation, dans un autre contexte (dans le cas indiqué, définir un autre contexte consiste à utiliser d'autres emplacements).

4.3. RESOLUTION DU CONTEXTE. EDITION DE LIEN.

Dans la technique interprétative, l'accès aux objets nécessite un calcul, mais les objets sont libres. Dans la technique d'assemblage, l'accès aux objets est immédiat ou résulte d'un simple adressage, mais les objets sont liés. Afin d'éviter les inconvénients de ces deux méthodes, on peut choisir un compromis.

Si l'identificateur désigne un objet connu, on le lie, c'est-à-dire qu'on procède par substitution

Dans le cas contraire, on réalise une mise en oeuvre plus rapide de la fonction "tabulée"

$$f(s) \xrightarrow{\text{fournit}} \text{objet du contexte}$$

en la transformant en une fonction d'indexation, suivie éventuellement d'un chaînage.

Soit $C = (\mathcal{L}, \mathcal{D})$ un contexte, et $\bar{\mathcal{L}} = \{s_0, s_1, \dots, s_p\}$ un ensemble où s est la chaîne de caractères représentant le j^e identificateur de $\mathcal{L}$. Les objets de $\mathcal{D}$ sont, à l'exécution, représentés par des noms ou des descripteurs inconnus au stade de la compilation (rappelons qu'une valeur entière est un particulier de descripteur; voir 3.22.).

1) Soit $N = \{n_0, n_1, \dots, n_p\}$ l'ensemble des noms représentant les objets de $\mathcal{D}$. On suppose que tous ces noms ont la propriété de contiguïté

$$n_j = n_{j-1} + 1 \quad , \text{ quel que soit } j = 1, \dots, p$$

On décompose alors la fonction $f(s_j) = n_j$ en deux fonctions

$$\begin{cases} f_1(s_j) = n_j & \text{de } \bar{\mathcal{L}} \text{ dans } \{0, 1, \dots, p\} \\ f_2(j) = n_0 + j & \text{de } \{0, 1, \dots, p\} \text{ dans } N \end{cases}$$

La fonction f_1 peut être mise en oeuvre au stade de la compilation. Pour mettre en oeuvre f_2 , au stade de l'exécution, il suffit de connaître n_0 et d'indexer sur cette valeur. n_0 est le nom de l'emplacement origine d'un ensemble d'emplacements contigus appelé région. Une région constitue tout ou partie d'un segment.

Cette mise en oeuvre peut être utilisée chaque fois que l'on met en place un contexte formé d'un ensemble de repères à créer.

2) Soit $N = \{n_0, n_1, \dots, n_p\}$ l'ensemble des noms représentant les objets de $\mathcal{D}$. On suppose que ces noms sont quelconques.

Soit $M = \{m_0, m_1, \dots, m_p\}$ un ensemble de noms désignant des emplacements contigus. On décompose alors la fonction $f(s_j) = n_j$ en trois fonctions:

$$\begin{cases} f_1(s_j) = j & \text{de } \bar{\mathcal{L}} \text{ dans } \{0, 1, \dots, p\} \\ f_2(j) = m_0 + j = m_j & \text{de } \{0, 1, \dots, p\} \text{ dans } M \\ f_3(m_j) = n_j & \text{de } M \text{ dans } N \end{cases}$$

La fonction f_1 est mise en oeuvre à la compilation. La fonction $f_3 \circ f_2$ est mise en oeuvre à l'exécution grâce à une indexation (f_2) suivie d'un chaînage (f_3). Chaque emplacement de nom m_j est appelé un lien. Leur ensemble forme une région de liens.

Cette mise en oeuvre est nécessaire chaque fois que l'on établit un contexte formé de repères déjà créés (exemple: passage de paramètres).

3) Soit $\mathcal{V} = \{v_0, v_1, \dots, v_p\}$ l'ensemble des descripteurs, inconnus, représentant les objets de $\mathcal{D}$. Dans ce cas, on considère aussi une région de liens $\mathcal{M} = \{m_0, m_1, \dots, m_p\}$, et $f(s_j) = n_j$ est représenté par:

$$\begin{cases} f_1(s_j) = j & \text{de } \mathcal{L} \text{ dans } \{0, 1, \dots, p\} \\ f_2(j) = m_j & \text{de } \{0, 1, \dots, p\} \text{ dans } \mathcal{M} \\ f_3(m_j) = v_j & \text{de } \mathcal{M} \text{ dans } \mathcal{V} \end{cases}$$

Cette mise en oeuvre est nécessaire chaque fois que l'on établit un contexte formé d'objets dont la représentation n'est calculable qu'à l'exécution.
(descripteur d'un tableau, valeur d'une expression).

4.31. Représentation d'un identificateur.

Grâce aux techniques présentées ci-dessus, on peut représenter, dans un objet-procédure, un identificateur par une représentation de fonction d'accès.

Cette fonction d'accès peut être:

a) l'identité. L'identificateur est représenté par la valeur v , connue, de l'objet désigné (on appelle souvent ce mode de désignation l'adressage immédiat).

b) une indexation. L'identificateur est représenté par un indice dans une région, soit i . La représentation doit avoir une seconde composante, si plusieurs régions peuvent exister simultanément, permettant de distinguer une région. Soit r cette composante. L'identificateur est représenté par le couple (r, i) . (Ce mode de désignation est appelé adressage indexé).

c) une indexation suivie d'un chaînage. L'identificateur est représenté par le couple (r, i) auquel il convient d'adjoindre une marque de chaînage (adressage indirect). On note cependant que ce chaînage peut être indiqué par un préfixe adéquat dans le lien de chaînage.

v , (r, i) ou $*(r, i)$ fournissent la représentation de l'objet désigné.

4.32. Mise en oeuvre de la fonction de désignation. Edition de lien.

Considérons, dans le couple (r, i) , la composante r . Pour mettre en oeuvre l'indexation, il suffit que r désigne un emplacement de nom connu, un registre par exemple. Ce registre contient, à l'exécution, le nom de la base d'une région. C'est un descripteur de région, qui réalise un chaînage sur la région.

On appelle édition de lien, l'opération permettant d'établir une relation de désignation (on lie l'identificateur ou sa représentation à un objet). Cette opération est effectuée à différents moments et de diverses façons.

Si elle est effectuée au stade de la compilation, l'objet est la représentation de l'identificateur. Si elle est effectuée au stade de l'exécution, sa mise en oeuvre nécessite de garnir un registre dans certains cas simples. Elle nécessite de garnir un registre et un ensemble de liens dans le cas général.

Le garnissage du registre ne pose aucun problème puisque la région qu'il décrit est créée au moment de l'établissement des relations de désignation. Le garnissage des liens nécessite de connaître la représentation de chaque objet désigné. Celle-ci sera connue car elle appartient par définition à une chaîne d'accès construite à partir d'un autre identificateur.

Remarques: 1) Dans de nombreux systèmes, on peut effectuer le garnissage d'un lien au stade dit de "l'édition de lien", qui précède l'exécution. L'ensemble des liens associés à une procédure s'appelle son "vecteur de transfert". Souvent, afin d'éviter le chaînage à l'exécution, on vient substituer, dans les instructions de la procédure, le contenu du lien à la représentation de la fonction d'accès à ce lien. Dans ces conditions, la procédure n'est pas un objet libre.

2) Un bloc de contrôle de fichier (DCB) est un lien qui contient à l'exécution un chaînage sur le fichier et la représentation, sous forme d'une chaîne de caractères, de l'identificateur de fichier. Dans certains systèmes, ce lien est un objet accessible, désigné explicitement par le programme, qui peut en changer le contenu.

4.4. ELABORATION D'UNE VALEUR.

Une valeur est dite élaborée lorsqu'elle peut être confiée, pour traitement, à un processeur. Une élaboration est donc relative à la nature d'un processeur. On cherche évidemment, dans un système, à ce qu'une valeur soit élaborable par un processeur câblé.

Le compilateur peut élaborer les valeurs primitives. Il ne peut élaborer la valeur d'un repère, ni celle d'un objet composé de taille variable. En ce qui concerne les procédures, leur élaboration n'est en général pas possible au stade de la compilation; en effet, pour qu'une procédure soit élaborée, il faut que toutes les relations de désignation entre les identificateurs globaux à la procédure et les objets désignés soient établies.

4.5. REPRESENTATION D'UNE PROCEDURE.

Une procédure, sous une forme libre, est un objet composé

- d'une suite d'instructions, chaque instruction étant un appel d'une procédure primitive avec un ou plusieurs paramètres. Chaque paramètre est une représentation d'un identificateur.
- d'une représentation idoine des contextes des objets que son exécution ne crée pas.

Nous verrons aux chapitres 5., 6., et 7., différentes formes de représentation possible.

Lorsque la procédure est liée, il faut lui adjoindre, pour chaque processus, la représentation de tous les contextes des objets auxquels elle a accès. Cette représentation utilise un ensemble de registres et de liens.

CHAPITRE 5 . LE SYSTEME CLICS

5.1. INTRODUCTION

Le système CLICS (Classroom Information and Computing Service) [1], est une présentation idéalisée à des fins pédagogiques du système MULTICS [2], lequel a été réalisé sur une machine HIS 645.

Dans CLICS, une session d'utilisateur est définie par l'intervalle de temps qui sépare l'appel de deux procédures *LOGIN* et *LOGOUT*. On appelle processus l'exécution de la "procédure" dont la notation est

LOGIN

<représentation de la suite d'actions effectuées par l'utilisateur>

LOGOUT

Il existe autant de processus que d'utilisateurs pouvant utiliser simultanément le système. On peut considérer que chacun de ces processus peut être dans deux états: en cours d'exécution (il est actif) ou en attente d'un événement qui le réveille (il est passif).

Le moniteur de CLICS est composé d'un ensemble de procédures partagées utilisables par tous les processus. Ce sont les procédures primitives de chaque processus. Chaque processus peut donc, à un instant donné, remplir des fonctions moniteur et à un autre instant exécuter des procédures écrites par un utilisateur. Les procédures *LOGIN* et *LOGOUT* commutent l'état d'un processus. *LOGIN* déclenche l'événement qui réveille le processus, *LOGOUT* déclenche l'événement qui le met en sommeil. La création d'un programme est consécutive à l'appel d'une procédure primitive d'identificateur *FORTTRAN* par exemple. Une telle procédure crée un ou plusieurs objets procédures qui sont catalogués. Outre la classe des objets procédure, il existe une seconde classe d'objets qu'un utilisateur peut créer et cataloguer: les objets collection de données. Ils sont partageables, pour la plupart, et sont représentés dans des segments. L'espace des segments constitue la mémoire segmentée: l'accès à chaque emplacement de cette mémoire est câblé.

Dans la suite de ce chapitre, nous présentons la mémoire segmentée et les différents objets qu'elle peut contenir, la façon dont un processus les

désigne, et enfin la façon de mettre en oeuvre les relations de désignation.

5.2. LA MEMOIRE SEGMENTEE

Un segment, dans CLICS, peut contenir de 1 à 2^{24} mots. Un descripteur de segment le situe dans une mémoire fictive (voir en 1.2.) de 2^{40} mots. La gestion des ressources physiques n'a donc pas à être considérée ici.

Le descripteur de segment comprend:

- la longueur du segment,
- l'adresse, en mémoire fictive, de la base du segment,
- des indicateurs utilisés par le mécanisme de protection.

Tout accès à un mot du segment fait intervenir le descripteur de segment.

Remarque: L'application des segments dans la mémoire fictive fait passer d'un ensemble de suites linéaires, les segments, dans une suite linéaire unique, la mémoire fictive. Cette transformation nécessite une gestion de la mémoire fictive analogue à la gestion des mémoires physiques. Cette gestion est faite dans CLICS à l'aide d'un dispositif de pagination. [1], [3], [4]

5.21. Désignation d'un segment par un processus. Le descriptif.

Tout segment **S**, accessible à un processus, est lié, à un moment donné de l'exécution, à son espace adressable. Dans ce but, on pourrait imaginer qu'un lien associé à toute procédure qui désigne **S** contienne le descripteur de **S**.

Cependant, un segment est partageable et chaque processus a seulement un certain pouvoir d'accès à ses emplacements. Un processus peut modifier les contenus de ses emplacements, un autre peut seulement lire ces contenus. Aussi, comme nous l'avons vu (chap. 3), il faut disposer d'un exemplaire de descripteur propre au processus. Dans CLICS, cet exemplaire est le triplet (adresse de base, longueur, pouvoir d'accès).

Le descriptif du processus est un segment contenant l'ensemble des descripteurs, propres au processus, et liés à ce processus. C'est l'indice de l'emplacement, dans le descriptif, d'un descripteur de segment, qui constitue le nom du segment pour ce processus.

Ce nom est évidemment propre au processus. En effet, soit deux processus qui désignent le même segment. Si on veut conserver un nom unique dans les deux processus, ce nom doit être réservé, ce qui veut dire que les noms des segments partagés sont attribués de façon statique dans chaque processus. Pour utiliser moins de noms et en particulier pour ne pas lier les segments non utilisés par un processus, les noms sont alloués dynamiquement à chaque processus. Il n'y a plus de raison pour que les segments partagés aient le même nom. (fig. 1)

Un nom d'un segment, propre à chaque processus, est un entier, compris entre 0 et $2^{16}-1$. Le nom i par lequel le processus $\mathcal{P}_1$ désigne le segment $\mathcal{S}$ correspond au $(i+1)^{\text{ème}}$ mot du descriptif du processus $\mathcal{P}_1$; si $\mathcal{P}_2$ désigne $\mathcal{S}$ du nom j , le descriptif de $\mathcal{P}_2$ aura pour $(j+1)^{\text{ème}}$ mot un descripteur de $\mathcal{S}$. Le descriptif a un descripteur particulier contenu dans un registre spécial, le registre de base du descriptif. (fig. 1)

5.22. Nom d'un mot de segment.

Chaque mot de segment $\mathcal{S}$ a pour nom un couple (nom du segment, indice dans le segment) le nom du segment étant propre au processus qui désigne le mot, l'indice dans le segment étant le numéro d'ordre du mot dans le segment considéré. Le $(d+1)^{\text{ème}}$ mot du segment $\mathcal{S}$ a donc pour nom (s_i, d) , quand le processus $\mathcal{P}_i$ y fait référence.

Le nom d'un mot de segment est une adresse segmentée. Le processeur traduit une adresse segmentée (s_i, d) de la manière suivante:

le contenu du registre de base du descriptif désigne le descriptif dont le $(s_i+1)^{\text{ème}}$ mot est le descripteur, pour $\mathcal{P}_i$, du segment $\mathcal{S}$. Le $(d+1)^{\text{ème}}$ mot dans ce segment est le mot recherché. (fig. 1)

5.23. Descriptif et processus.

Par construction, les registres et le descriptif représentent l'espace adressable du processus. Le descriptif n'a pas de descripteur dans le descriptif, et il n'est pas connu par les autres processus.

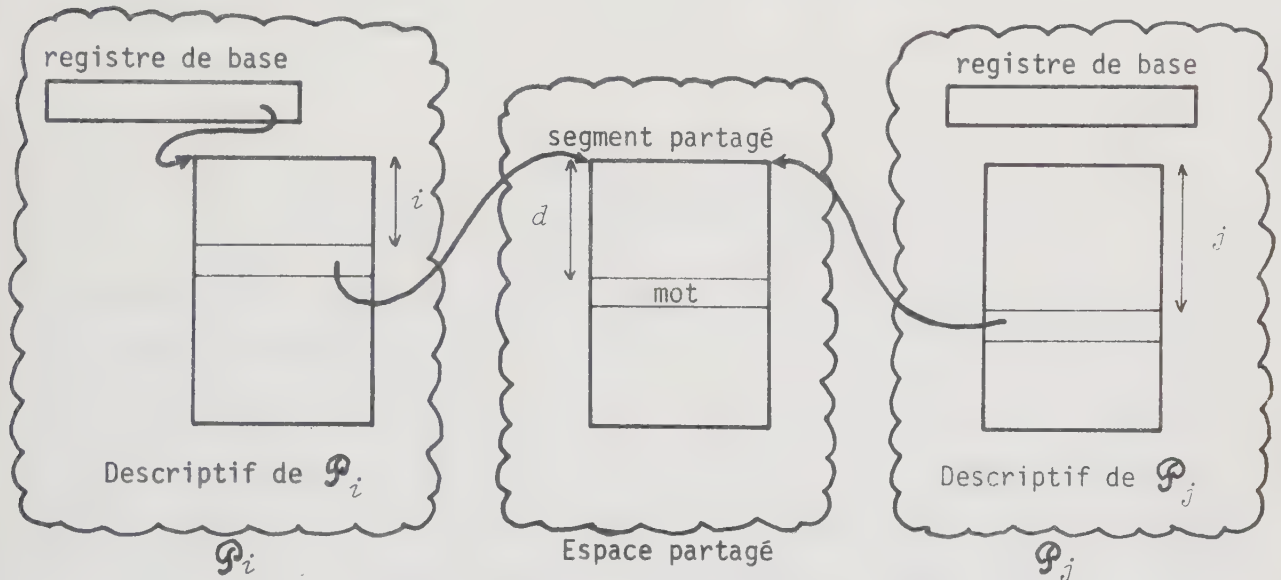


fig. 1. Partage d'un segment

5.3. LANGUAGE MACHINE ET OBJETS MANIPULES.

Dans ce qui suit, nous observons un processus à un instant donné de son évolution, et nous cherchons à montrer comment il accède aux objets (données, procédures) qu'il manipule. L'utilisateur désigne par un identificateur les objets qu'il crée ou ceux que les autres créent. Nous verrons au paragraphe 5.5. quand et comment la relation de désignation est obtenue. Auparavant, afin d'illustrer les mécanismes d'accès, indiquons le format des instructions de la machine, les objets que l'on veut manipuler et les problèmes d'adressage qui en résultent.

5.31. Format des instructions.

Le langage machine comprend des instructions de longueur fixe, contenues

dans un mot. On considère ici que l'objet élémentaire que peut désigner un processus est représentable par un mot de segment. Les instructions sont généralement composées de 3 champs:

- le code-opération,
- le nom d'un registre (éventuellement). Chaque processus utilise 16 registres généraux notés $R0, R1, \dots, R15$. Les registres $R0, R3, R4$ et $R5$ ont une fonction très particulière que nous expliciterons.
- un champ adresse.

5.32. Les différents types d'objets manipulés.

Considérons une procédure $\mathcal{P}$. On répartit en 3 classes les objets qu'une procédure peut désigner.

5.321. Les fichiers.

Si on fait abstraction des problèmes de restrictions d'accès, tous les fichiers de la communauté (y compris l'objet-procédure $\mathcal{P}$) sont accessibles depuis $\mathcal{P}$ avec les règles suivantes.

- Si le fichier est une collection de données, il est représenté dans un segment appelé segment de données. Un constituant élémentaire du fichier est représenté par un mot du segment renfermant sa valeur primitive, ou contenant la valeur primitive qu'il repère. La fonction d'accès est une indexation par rapport au premier mot du segment. (voir cependant 5.55)
- Si le fichier est une procédure, il est représenté dans un segment appelé segment-procédure. Nous étudierons ultérieurement sa représentation. Si le fichier est un module, il est aussi représenté dans un segment que nous appelons segment-procédure. Dans ce cas, on peut désigner certaines des procédures du module, appelées points d'entrée.

La durée de vie de ces objets est déterminée par des opérations explicites de création et de destruction. Ces objets, dits objets connus du système, sont décrits dans un catalogue. A chaque fichier est associée une fonction d'accès se présentant comme une suite d'identificateurs désignant les

noeuds d'une arborescence. Cette fonction fournit un repère de fichier quand le processus a un pouvoir adéquat (il peut alors modifier l'objet repéré, le détruire, etc...). Elle fournit le fichier lorsque le processus veut lier celui-ci à son espace adressable: ceci se traduit par l'obtention d'un exemplaire de descripteur avec des restrictions d'accès aux constituants, variables en fonction du processus demandeur. Nous nous plaçons ici dans cette seconde situation et nous supposons que la fonction d'accès se réduit à la donnée d'un seul identificateur.

5.322. Les objets propres à la procédure.

Une procédure peut désigner, et elle seule, des objets propres. On appelle valeur élémentaire une valeur de type primitif représentable sur un mot.

- Les objets rémanents. Ce sont des repères de valeur élémentaire, créés au moment où la procédure est liée à un processus. Ils sont détruits lorsque le processus se termine. Leur valeur est une adresse segmentée.

- Les objets locaux. Ce sont des repères de valeur élémentaire, créés par la procédure chaque fois qu'elle est appelée. Ils sont détruits lorsque l'exécution de la procédure est terminée. Leur valeur est aussi une adresse segmentée.

- Les composants de la procédure. Ce sont tous des objets constants - instructions, valeurs élémentaires - rangés dans les emplacements du segment-procédure. Leur descripteur est une adresse segmentée qui réalise un chaînage sur l'emplacement renfermant leur valeur.

Les objets rémanents et locaux repèrent à leur création une valeur initiale élémentaire fixée à la compilation. Pour caractériser le rôle de ces objets, on note que, à chaque appel de $\mathfrak{P}$ par le processus, $\mathfrak{P}$ trouve les objets rémanents dans l'état d'exécution que leur avait donné l'exécution précédente et crée de nouveaux objets locaux repérant une valeur initiale donnée.

5.323. Les paramètres effectifs.

Ce sont des objets propres à la procédure appelante, des paramètres effectifs de la procédure appelante ou un fichier désigné par la procédure appelante.

5.33. Problèmes liés à l'adressage des objets d'une procédure.

Les problèmes d'adressage des objets accessibles proviennent de deux objectifs du système:

- chaque procédure et chaque collection de données sont potentiellement partageables;
- les procédures peuvent s'appeler récursivement.

Ces caractéristiques impliquent les conséquences suivantes.

- a) Les fichiers, qu'il s'agisse de procédures ou de collections de données, ne doivent pas être dupliqués pour chaque processus.
- b) L'utilisation que fait un processus $\mathcal{P}_i$ d'un segment-procédure $\mathcal{P}$ ne doit pas contrarier l'utilisation qu'en fait simultanément le processus $\mathcal{P}_j$. Par contre l'utilisation par $\mathcal{P}_i$ d'un segment partagé représentant un ensemble de repères peut interférer avec l'utilisation qu'en fait le processus $\mathcal{P}_j$.
- c) Les objets dont la durée de vie est égale à celle du processus (registres et objets rémanents) figurent en un exemplaire par processus. En effet, l'utilisation que fait d'un tel objet le processus $\mathcal{P}_i$ ne doit pas interférer avec l'utilisation qu'en fait le processus $\mathcal{P}_j$.
- d) Les objets dont la durée de vie est égale à celle de la procédure (locaux) figurent en un exemplaire par appel de procédure et par processus. Autrement dit, pour chaque processus $\mathcal{P}_i$, il doit y avoir autant d'exemplaires d'un tel objet que d'exécutions en cours de la procédure qui l'utilise. Une pile est nécessaire pour gérer de tels objets.

Le paragraphe suivant explicite la manière de désigner, dans une instruction, ces différents types d'objets.

5.4. DESIGNATION DES OBJETS.

Considérons un segment S désigné par les processus $\mathcal{P}_i$ et $\mathcal{P}_j$ respectivement sous les noms s_i et s_j . Considérons un emplacement de numéro d dans le segment S . Il a pour noms respectifs (s_i, d) pour $\mathcal{P}_i$ et (s_j, d) pour $\mathcal{P}_j$. Aucune instruction ne peut donc avoir en opérande un nom pour désigner un emplacement quelconque d'un segment partagé (même si l'instruction est contenue dans ce segment). Pour ce faire, on utilise une indexation.

Il existe deux types d'indexation: l'indexation partielle (notée ip dans le champ adresse) et l'indexation composée (notée ic). Le champ adresse d'une instruction est ainsi formé:

- une marque d'indexation (ip ou ic),
- un couple (nom de registre, déplacement), soit (r, dep) .

Si le registre r contient l'adresse segmentée (s, d) , alors le nom (adresse segmentée) du mot recherché est respectivement:

- (s, dep) si l'indexation est partielle,
- $(s, dep+d)$ si l'indexation est composée.

5.41. Désignation des composants de la procédure.

Les instructions de branchement et les instructions de référence aux valeurs élémentaires (qui ne peuvent être représentées dans le champ adresse de l'instruction) utilisent l'adressage indexé partiel. Le registre RO contient à tout moment le nom par lequel le processus désigne le segment-procédure en cours d'exécution. Ainsi, on désigne les composants du segment-procédure par le triplet: (ip, RO, d) , où d a pour valeur le déplacement par rapport au début du segment-procédure.

La partie déplacement de RO du processus $\mathcal{P}_i$ est à tout instant l'indice (dans le segment) de l'instruction en cours. RO est un compteur ordinal.

5.42. Désignation des objets rémanents.

Rappelons que les objets rémanents ont la durée de vie du processus et qu'il doit y en avoir un exemplaire par processus. Ils ne peuvent être rangés dans le segment-procédure. Par contre, comme leur valeur initiale est identique pour tous les processus, elle est rangée dans une zone du segment-procédure.

Lorsqu'un segment-procédure S est lié à un processus P_i , un nouveau segment est créé et est inclus dans le descriptif de P_i . C'est le segment de liaison, $\mathcal{L}(S, i)$, associé au segment-procédure S . Les valeurs initiales des objets rémanents sont recopiées en tête de ce segment de liaison.

Notons que:

- ce segment est inaccessible aux autres processus;
- ce segment est créé une seule fois et réutilisé lors de toute nouvelle exécution du segment-procédure par le processus considéré;
- chaque processus possède autant de segments de liaison qu'il connaît de segments-procédures différents.

En conséquence, les opérations de S sur des objets rémanents désignent des emplacements du segment de liaison.

Soit le segment S utilisé par deux processus P_i et P_j , $\mathcal{L}_i$ est le nom de $\mathcal{L}(S, i)$ pour P_i , $\mathcal{L}_j$ est le nom de $\mathcal{L}(S, j)$ pour P_j . La référence aux objets rémanents s'effectue par indexation partielle, en utilisant un registre général, appelé registre pointeur de liaison. Par convention de programmation, le registre $R4$ est utilisé. Une opération désignant un objet rémanent a la forme suivante:

opération	($\mathcal{L}_i$, $R4$, déplacement)
-----------	---

5.43. Désignation des fichiers.

Les fichiers de la communauté des utilisateurs sont représentés dans des segments et chacun de leurs composants est considéré au même titre qu'un objet propre à une procédure. Pour désigner un tel composant, on doit

évidemment utiliser un lien de chaînage (chap. 4).

Considérons deux processus $\mathcal{P}_i$ et $\mathcal{P}_j$ utilisant le segment-procédure $\mathcal{S}$, lequel désigne l'emplacement d du segment $\mathcal{S}_1$, connu par $\mathcal{P}_i$ et $\mathcal{P}_j$ sous les noms s_i et s_j . Un mot du segment de liaison contient le nom d'un emplacement de $\mathcal{S}_1$, soit

- (s_i, d) dans $\mathcal{L}(\mathcal{S}, i)$
- (s_j, d) dans $\mathcal{L}(\mathcal{S}, j)$.

Une opération désignant un tel emplacement utilise une indirection et une indexation partielle.

opération * ($ip, R4$, déplacement dans le segment de liaison)

L'indirection (notée *) consiste à calculer, grâce à l'indexation, l'adresse segmentée d'un emplacement qui contient l'adresse segmentée de l'objet référence. Nous verrons ultérieurement à quel moment une adresse segmentée est rangée dans le segment de liaison et par quelle méthode (5.5.).

5.44. Désignation des objets locaux.

Les objets locaux ont pour durée de vie la durée d'exécution de la procédure. Appelons région des locaux l'ensemble d'emplacements qu'il est nécessaire de créer pour les représenter. Par le jeu des appels de procédure, des régions successivement créées peuvent avoir des durées de vie imbriquées. On gère de telles régions en les rassemblant dans une pile.

Pour chaque processus, un segment particulier est géré en pile. Le registre $R3$, appelé aussi registre pointeur de pile, contient une adresse segmentée qui désigne constamment la base de la dernière région empilée. Cette région est celle de la procédure en cours d'exécution. Par ailleurs, le premier mot de la pile contient toujours l'adresse segmentée du sommet de pile.

Considérons un emplacement d'une région. Soit n son indice dans la région. Soit p_i et p_j les noms des piles associées respectivement à $\mathcal{P}_i$ et $\mathcal{P}_j$. Dans une procédure, une instruction désigne un emplacement représentant

un objet local grâce à une indexation composée et le numéro n de l'emplacement. On a une instruction de la forme

opération $(\underline{ic}, R3, n)$

Elle désigne l'emplacement de rang $n+1$ de la dernière région empilée, qu'elle soit exécutée par le processus $\mathcal{P}_i$ ou par le processus $\mathcal{P}_j$.

5.45. Désignation des paramètres effectifs.

Rappelons que la durée de vie d'un paramètre effectif est au moins égale à celle de l'exécution de la procédure appelante. La procédure appelante connaît son nom en fonction de sa nature.

- Objet fichier: son nom est rangé dans le segment de liaison de la procédure appelante. C'est une adresse segmentée $(s, d1)$.
- Objet rémanent de la procédure appelante: son nom est une adresse segmentée $(l, d2)$, déduite du couple $(R4, d2)$ et du nom de segment l contenu dans $R4$.
- Objet local de la procédure appelante: son nom est une adresse segmentée $(p, d3 + d4)$ déduite du couple $(R3, d4)$ et du nom de région $(p, d3)$ contenu dans $R3$.
- Paramètre de la procédure appelante: nous verrons plus loin comment obtenir son nom.

En toute généralité, l'objet désigné par le paramètre effectif est un repère ou un élément de fichier. Dans la procédure appelée, on cherche donc à désigner un emplacement existant ayant déjà un nom. On va, pour cela, utiliser un lien (voir chap. 4). Pour chaque paramètre effectif désigné, la procédure appelante garnit un lien de la valeur de son nom. On appelle région des liens formel/ effectif un ensemble de tels liens. Cette région, par le jeu des appels de procédure, doit être empilée pour la durée d'exécution de la procédure.

Dans la procédure appelée, pour désigner l'objet effectif, on utilise un chaînage utilisant le contenu du lien. Le contenu d'un registre, le registre $R5$, désigne la base de la région des liens formel/effectif. Une instruction de la procédure désigne un paramètre effectif grâce à une indexation composée suivie d'une indirection. On a

opération $*(\underline{ic}, R5, i)$

5.46. Illustration des mécanismes de désignation.

Rappelons l'utilisation des registres du processus:

$R0$: segment-procédure en cours d'exécution

$R3$: base de la région des locaux

$R4$: segment de liaison associé à la procédure

$R5$: base de la région des liens formel/effectif

et le mode de désignation des objets:

- objet fichier $* (\underline{ip}, R4, d)$
- objet rémanent $(\underline{ip}, R4, d)$
- objet local $(\underline{ic}, R3, d)$
- paramètre $* (\underline{ic}, R5, d)$
- valeur interne $(\underline{ip}, R0, d)$

La donnée d'un quadruplet $x(\underline{y}, reg, d)$ à un processeur fournit la valeur de l'objet désigné grâce à un chaînage éventuel (si $x \equiv *$). Cette valeur est une adresse segmentée, c'est-à-dire le nom d'un mot.

Dans l'illustration de la figure 2, nous représentons un processus $\mathcal{P}$ en train d'exécuter un segment-procédure $\mathcal{S}$ dans lequel figure tout l'éventail des désignations.

5.5. ETABLISSEMENT DES RELATIONS DE DESIGNATION.

Nous examinons maintenant comment et quand une procédure établit les relations de désignation entre les identificateurs qu'elle utilise et les objets désignés. Nous en déduisons alors la représentation d'une procédure partageable, sous sa forme libre (telle qu'elle est conservée dans le système) et sous sa forme liée (lorsqu'elle a été élaborée par un processus).

Aucune relation de désignation avec les objets de type repère et avec les fichiers ne peut être établie au stade de la compilation: l'adresse segmentée qui est la valeur d'un repère n'est déterminable qu'à l'exécution. Aucune relation de désignation avec les valeurs élémentaires autres que les valeurs immédiates ne peut être établie au stade de la compilation pour la même raison, puisqu'on utilise un chaînage (descripteur) pour les représenter.

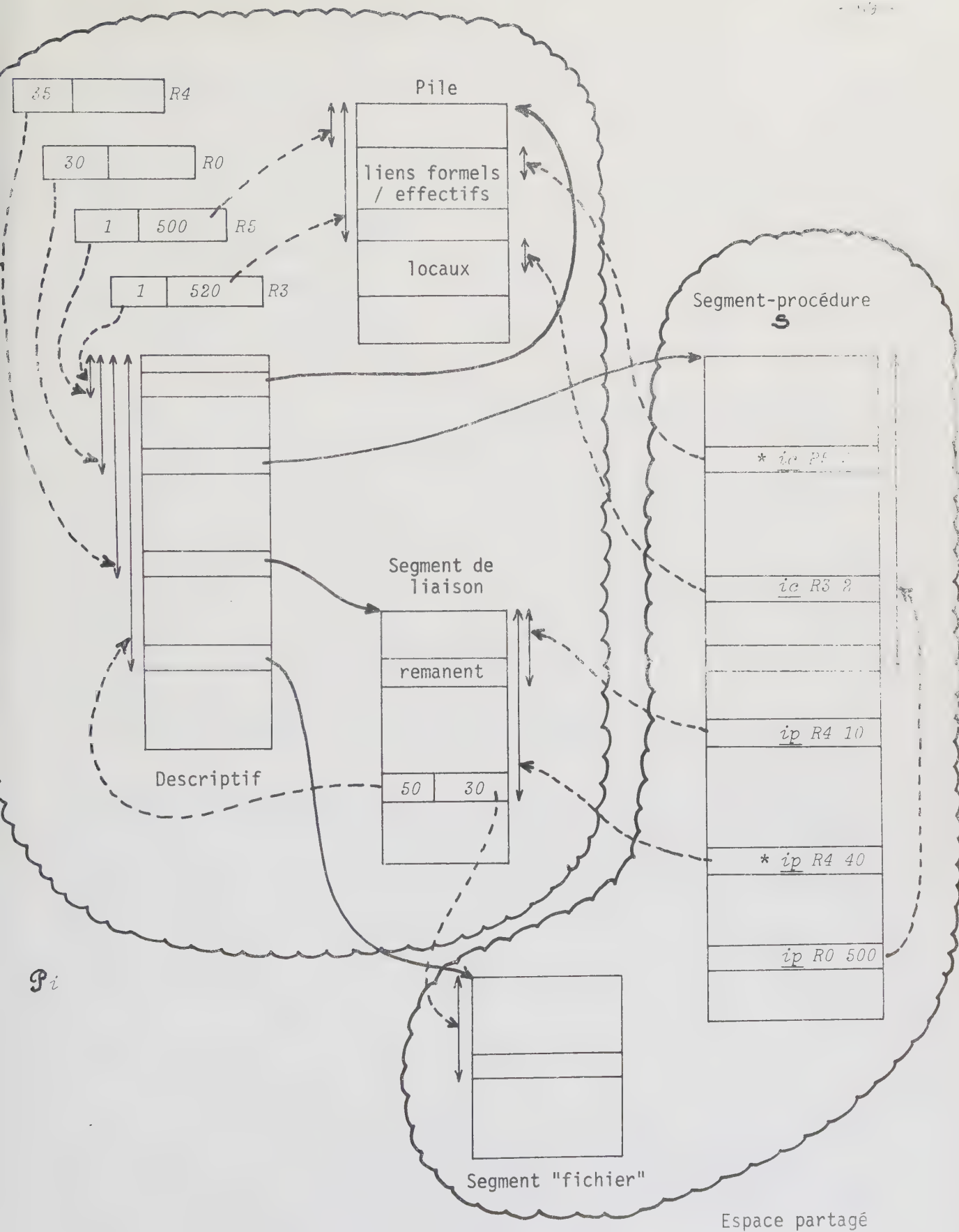


Fig. 2. Illustration des mécanismes de désignation.

5.51. Objets internes, locaux et rémanents.

Considérons les trois lexiques suivants:

- $\mathcal{L}_i$: lexique des identificateurs désignant des valeurs élémentaires ou des instructions (étiquette).
- $\mathcal{L}_l$: lexique des identificateurs désignant les repères locaux (repères de valeurs élémentaires).
- $\mathcal{L}_r$: lexique des identificateurs désignant les repères rémanents (repères de valeurs élémentaires).

Les identificateurs des trois lexiques désignent des objets respectivement créés:

- à la compilation (objets désignés par $\mathcal{L}_i$),
- à l'exécution, au moment de l'établissement de la relation de désignation et pour la durée de celle-ci (objets désignés par $\mathcal{L}_l$ et $\mathcal{L}_r$).

Par ailleurs, aucun de ces objets n'est accessible d'une autre procédure par l'identificateur qui le désigne. Aussi, dans ces trois cas de figures, il n'y a aucune difficulté à considérer trois régions d'emplacements contigus pour représenter les objets. Ces régions sont respectivement:

- le segment-procédure (objets désignés par $\mathcal{L}_i$),
- la région des locaux (objets désignés par $\mathcal{L}_l$),
- la région des rémanents (objets désignés par $\mathcal{L}_r$).

Le lexique $\mathcal{L}_l$ est un ensemble mis en relation biunivoque avec l'ensemble ordonné d'emplacements que constitue la région des locaux. Il en est de même entre $\mathcal{L}_r$ et la région des rémanents. Entre le lexique $\mathcal{L}_i$ et le segment-procédure, il existe une relation injective puisque tous les emplacements ne sont pas désignés par un identificateur.

Ainsi la relation de désignation

$$I \xrightarrow{\text{désigne}} (s, d)$$

se représente par une fonction d'accès simple

$$(reg, i) \xrightarrow{\text{fournit}} (s, b + i)$$

dont la composante statique i désigne un emplacement dans un ensemble et dont la composante reg est une désignation indirecte de l'ensemble de référence. reg est la désignation du registre qui contient, pendant toute la durée de la relation de désignation, l'adresse segmentée (s, b) .

Nous avons vu que les locaux et les rémanents repèrent dès leur création une valeur initiale connue à la compilation. Ces valeurs élémentaires doivent donc être représentées comme des valeurs internes. Pour accélérer l'établissement de ces relations de repérage, on crée deux maquettes de valeurs dans des emplacements contigus du segment-procédure: la maquette des rémanents et celle des locaux. Ainsi l'état d'exécution initial des trois contextes créés sur les lexiques $\mathcal{L}_i$, $\mathcal{L}_r$ et $\mathcal{L}_l$ est représenté dans le segment-procédure (on note que l'état d'exécution de $\mathcal{L}_i$ est constant; il se réduit au contexte puisque celui-ci ne contient pas de valeurs de type repère).

5.52. Paramètres.

Considérons le lexique $\mathcal{L}_p$ des identificateurs de paramètres formels. Par définition, chaque identificateur désigne, au cours de l'exécution de la procédure, un objet connu de la procédure appelante de type repère de valeur élémentaire ou fichier. Lorsque la procédure est appelée, ces objets sont désignés par une adresse segmentée. Ces objets n'étant pas groupés, il est nécessaire de créer un lien pour mettre en oeuvre la fonction d'accès (chap. 4).

Chaque élément du contexte construit sur ce lexique, soit $(F, (s, d))$, où (s, d) est l'adresse segmentée du paramètre effectif, se présente donc à l'exécution sous la forme

- d'une fonction d'accès (reg, i) qui fournit le nom d'un lien,
- d'un chainage par l'intermédiaire du lien sur l'adresse segmentée (s, d) .

La présence du chainage est indiquée par une marque d'indirection accompagnant le couple (reg, i) .

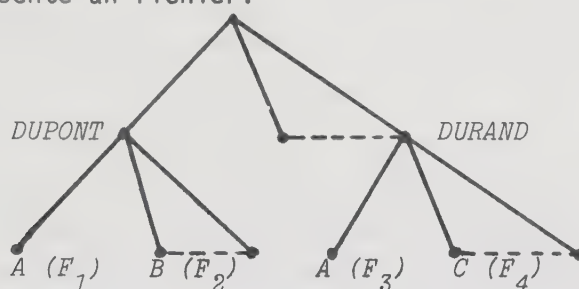
5.53. Fichiers.

Considérons le lexique $\mathcal{L}_f$ des identificateurs de fichiers. Chaque identificateur désigne un fichier représenté par un segment. Cette relation de désignation est immuable aussi longtemps que le fichier existe. Ainsi, contrairement aux liaisons paramètres formels / paramètres effectifs qui doivent être établies à chaque appel de procédure, la relation identificateur de fichier / objet désigné peut être établie une fois pour toutes.

Un fichier est un objet créé et détruit par les processus qui en ont les pouvoirs respectifs. Les moments de création et de destruction sont quelconques.

En outre, l'identificateur d'un fichier est en réalité une fonction d'accès qui se présente sous la forme d'une suite d'identificateurs. Cette fonction d'accès peut être complètement déterminée par cette suite ou au contraire variable, comme le montre l'exemple suivant.

Exemple: Considérons un catalogue simplifié, représenté comme une arborescence à deux niveaux (tout chemin est constitué de deux arcs). Chaque noeud de l'arbre est désigné par un identificateur et représente un utilisateur. Chaque feuille est désignée par un identificateur et représente un fichier.



La fonction d'accès *DUPONT.A* fournit le fichier F_1 . Par contre, la fonction d'accès *.A* fournit le fichier F_1 lorsque l'utilisateur *DUPONT* l'utilise, F_3 si c'est l'utilisateur *DURAND*.

Supposons que l'on fasse abstraction de la commodité introduite consistant à ne désigner un segment que par un nom dans un descriptif plutôt que par son descripteur. Il existe alors quatre raisons qui interdisent de ranger ce descripteur dans un segment-procédure et donc d'établir une relation de désignation à la compilation.

- 1) Le fichier désigné peut ne pas encore exister.
- 2) S'il existe, on ne sait pas forcément déterminer quel est le fichier désigné (fonction d'accès dynamique).
- 3) Si on connaît le fichier désigné (fonction d'accès statique), on ne connaît pas les restrictions d'accès le concernant puisqu'elles changent selon l'utilisateur qui utilise la procédure.

4) Si ces restrictions d'accès sont constantes, le fichier aura pu être détruit ou aura pu changer de taille au moment de l'exécution de la procédure.

Considérons le moment où la procédure est liée à un processus. Nous avons vu que c'est le moment choisi pour établir les relations avec les objets rémanents. A ce stade de l'exécution, on ne peut encore établir une relation avec un fichier pour la seule raison 1) énoncée ci-dessus. Le fichier peut ne pas exister si au cours de cette exécution particulière de la procédure il n'est pas désigné, ou si c'est justement l'exécution de la procédure qui va amener sa création.

La solution adoptée dans CLICS est d'établir la relation lors de la première désignation du fichier par le processus. On appelle cette opération: édition de lien dynamique. Elle est caractérisée par une élaboration incomplète de la valeur de type-procédure au moment de sa liaison, complétée au fur et à mesure des besoins pendant son exécution.

Comme pour les paramètres, on prépare un chaînage grâce à un lien. Au contraire des paramètres, ce lien n'est pas garni préalablement. Le processeur détecte que le chaînage n'est pas effectué grâce à un indicateur de déroutement que nous appelons lien non résolu. Au premier accès qui fait intervenir un tel lien, l'indicateur provoque l'appel d'une procédure d'édition de lien qui garnit le lien et efface l'indicateur de lien non résolu. Après quoi l'instruction qui désigne ce lien est réexécutée de nouveau: le mécanisme d'accès est alors semblable à celui qui est mis en oeuvre pour les paramètres.

Soit **S** la procédure qui désigne un fichier **S'**. L'indicateur de lien non résolu est rangé dans un lien associé à **S**. Il ne peut en effet être rangé dans l'instruction qui désigne ce lien, car cette instruction est à valeur constante puisque partagée.

Avant l'édition de lien, le lien doit contenir l'indicateur de lien non résolu et la chaîne de caractère déduite de l'identificateur. Cependant, cette chaîne doit également être conservée dans le segment-procédure pour toute liaison ultérieure effectuée par cette procédure pour le compte d'un autre processus.

Aussi, on n'en conserve qu'un seul exemplaire dans le segment-procédure, dans un emplacement de rang i , et le lien contient ce numéro (fig. 3.).

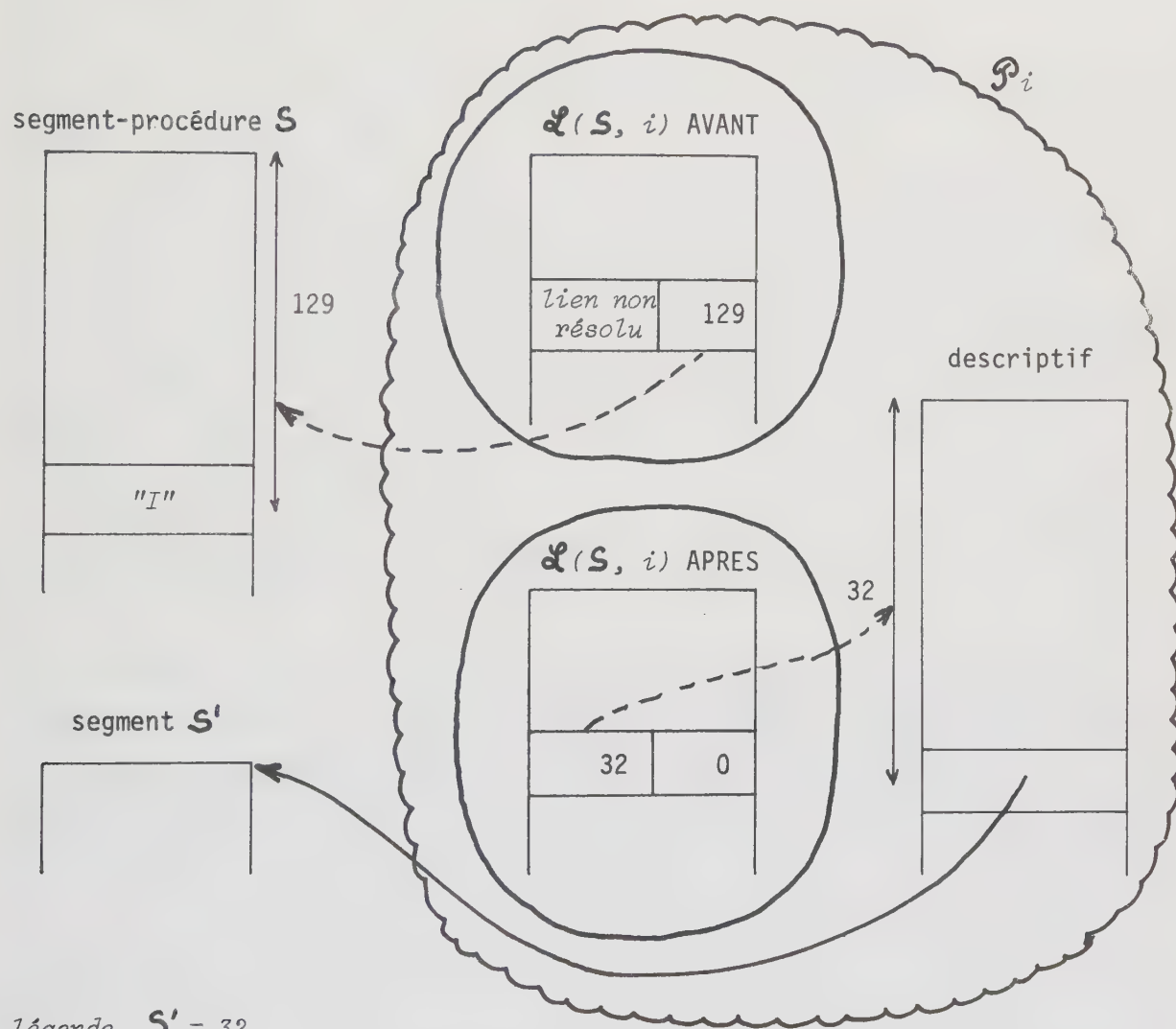


Fig. 3. Edition de lien dynamique.

5.54. Représentation des fichiers.

5.541. Procédure.

Nous avons vu les éléments qui entrent en jeu dans la représentation d'une procédure. On trouve, dans les emplacements d'un segment-procédure, successivement les éléments suivants:

- 1° la chaîne de caractère "p" déduite de l'identificateur P de la procédure et le numéro d'emplacement contenant la première instruction à exécuter;
- 2° la taille du segment de liaison (taille des deux maquettes suivantes);
- 3° la maquette des rémanents (valeur initiale repérée par chaque objet rémanent);
- 4° la maquette des liens de fichiers (couple (lien non résolu, j), j étant le numéro de l'emplacement renfermant la chaîne de caractères déduite de la fonction d'accès à un fichier);
- 5° les instructions de la procédure;
- 6° l'ensemble des emplacements renfermant les chaînes de caractères déduites de fonctions d'accès à des fichiers.

On note que le nombre de paramètres et de locaux n'est pas notée dans ces éléments. Le nombre de paramètres effectifs est rangé à l'appel de procédure par la procédure appelante dans la région des liens formel/effectif. La procédure appelée s'assure que ce nombre est adéquat. Les locaux sont réservés par les premières instructions de la procédure.

5.542. Module.

Un module est un ensemble de procédures ayant en commun des objets rémanents. Les références faites par le module aux fichiers sont l'union des références faites par chaque procédure. Dans la représentation du module dans un segment-procédure, on trouve successivement les éléments suivants:

- 1° une table de points d'entrée; chaque entrée dans la table est un couple (chaîne de caractères, numéro de la première instruction);
- 2° 3° 4° les mêmes éléments que ci-dessus (5.541);

- 5° les instructions des procédures constitutantes;
- 6° les mêmes éléments que ci-dessus (5.541).

On note que chaque procédure constitutive a ses propres paramètres formels et ses propres objets locaux.

5.543. Collection de données.

Une collection de données est un ensemble d'objets de type repère ou de type élémentaire ou encore un ensemble de tels ensembles. Une collection de données ne contient donc pas de désignation d'autres collections de données. Sa représentation contient les éléments suivants:

- 1° une table éventuellement vide, de points d'entrée; chaque entrée dans la table est un couple (chaîne de caractères, numéro du premier emplacement d'un ensemble d'emplacements);
- 2° 3° 4° aucun élément;
- 5° les emplacements de l'ensemble ou des ensembles;
- 6° rien.

5.55. Fonction d'accès à un fichier.

La fonction d'accès à un fichier se présente sous la forme de la suite d'identificateurs qui fournit le fichier, éventuellement postfixée par un identificateur. Soit *DUPONT.A* la fonction d'accès à un fichier. *DUPONT.A.E1* est la fonction d'accès à une procédure de ce fichier s'il est un module, à un ensemble d'objets de ce fichier s'il est une collection de données. C'est l'éditeur de lien qui vérifie qu'une telle désignation existe et qui détermine l'adresse segmentée correspondante.

Si le fichier est une collection de données, un tel accès permet de désigner un ensemble d'emplacements. Pour accéder à chaque composant de cet ensemble, on définit un type formel d'objet composé: un vecteur *T* de 10 éléments, une structure de 10 champs désignés par *C1*, *C2*, ... *C10*. La fonction d'accès complète s'écrit alors respectivement:

"DUPONT.A.E1" $p \rightarrow T(4)$

ou "DUPONT.A.E1" $p \rightarrow C4$

C'est le compilateur qui, grâce au type formel, a généré le calcul de l'accès au 4ème élément du sous-ensemble désigné par *DUPONT.A.E1*.

5.6. OPERATIONS SUR LES PROCEDURES.

Entre le moment où un processus désigne une procédure $\mathcal{P}$ et le moment où il se termine, on peut distinguer plusieurs états de la procédure, correspondant à diverses situations.

1) Liaison de la procédure.

Lorsque la procédure est liée à un processus, on range son descripteur dans le descriptif et on construit un segment de liaison. Ce faisant, on élabore les relations de la procédure avec ses composants internes (on garnit *R0* du nom du segment-procédure pour ce processus) et avec ses objets rémanents (on garnit *R4* du nom du segment de liaison). Une partie du contexte de la procédure est établie.

2) Appel de la procédure.

Si la procédure a été liée, c'est parce qu'elle était appelée par une procédure $\mathcal{Q}$. A chacun de ses appels par $\mathcal{Q}$, on établit le contexte des paramètres et le contexte local en garnissant les registres *R5* et *R3*.

Si la procédure est appelée par une autre procédure qui ne l'a pas encore appelée, l'éditeur de lien doit reconnaître qu'elle est déjà liée (elle a un nom) et retrouver le nom de son segment de liaison.

3) Accès aux fichiers.

Au fur et à mesure qu'elle s'exécute, la procédure établit des liens avec des fichiers, complétant s'il y a lieu l'établissement de son contexte particulier.

4) Appel d'une autre procédure.

Au cours de son exécution, la procédure peut appeler d'autres procédures, y compris elle-même. Afin de sauvegarder le contexte de la procédure $\mathcal{P}$, il faut sauvegarder le contenu des registres $R0$, $R3$, $R4$ et $R5$ et celui des autres registres généraux s'il y a lieu. Au retour dans $\mathcal{P}$, grâce à cette sauvegarde, on retrouve le contexte de $\mathcal{P}$ dont l'état d'exécution associé a éventuellement changé. Ces opérations de sauvegarde et de restauration s'effectuent dans la pile.

5.7. L'EDITEUR DE LIEN.

Au premier appel de $\mathcal{P}$ par la procédure $\mathcal{R}$ exécutée par le processus $\mathcal{P}_i$, l'éditeur de lien est mis en oeuvre. $\mathcal{P}$ est alors désigné par une chaîne de caractères, rangée dans $\mathcal{R}$, et retrouvée grâce au contenu du lien qui a provoqué le déroutement. Si c'est le premier appel de $\mathcal{P}$ par $\mathcal{P}_i$, l'éditeur de lien trouve dans le catalogue général son descripteur qu'il inclut dans le descriptif de $\mathcal{P}_i$, crée et initialise un segment de liaison $\mathcal{L}(\mathcal{P}, i)$. Il range le nom de $\mathcal{L}(\mathcal{P}, i)$ dans un segment propre à $\mathcal{P}_i$, appelé "table des pointeurs sur les segments de liaison"; cette table donne pour chaque nom de segment représentant un fichier lié à $\mathcal{P}_i$, le nom de ce segment de liaison. Il range, en outre, la chaîne de caractères représentant la fonction d'accès à $\mathcal{P}$ dans un segment propre à $\mathcal{P}_i$, appelé "catalogue des segments connus". Ce rangement s'effectue dans le $j^{\text{ème}}$ emplacement de la table si j est le nom de $\mathcal{P}$ pour $\mathcal{P}_i$.

Lors des appels ultérieurs de $\mathcal{P}$, l'éditeur de lien peut, grâce à ces tables, savoir que $\mathcal{P}$ est lié au processus et retrouver son nom et celui de son segment de liaison.

BIBLIOGRAPHIE

- [1] CLARK D., GRAHAM R., SALTZER J., SCHROEDER M.
The Classroom Information and Computing Service
MAC TR.80 . MIT Press (1971)

- [2] BENSOUSSAN A., CLINGEN C.T., DALEY R.C.
The Multics Virtual Memory : Concepts and Design
CACM 15, 7 (1972)

- [3] CROCUS (Ed.)
Les Systèmes d'Exploitation des Calculateurs. Principes de Conception
par un groupe de rédaction
A paraître (Dunod)

- [4] VERJUS J.P.
Quelques Propriétés des Langages d'Utilisation de Systèmes
Ecole d'été AFCET - Grenade (1973)

CHAPITRE 6 . LE SYSTEME BURROUGHS B 6700

6.1. INTRODUCTION.

Le système BURROUGHS B 6500/6700 [5,6,7,8] se particularise par les caractéristiques suivantes:

- marquage de tous les éléments de la mémoire par un préfixe,
- utilisation systématique de piles,
- partage des espaces adressables,
- création et destruction dynamique de processus.

Contrairement à CLICS, le système BURROUGHS B / 6700 ne permet pas l'édition de liens dynamique des fichiers.

Le moniteur est composé d'un ensemble de procédures partagées connues de tous les processus et appelées par ceux-ci en cas de besoin.

Dans ce chapitre, nous présentons la pile associée à un processus et les objets qu'elle contient, les noms qui les désignent, leur partage entre processus grâce à une pile arborescente et l'utilisation de celle-ci pour la gestion de l'ensemble du système.

6.2. LE MATERIEL.

6.21. Notion de préfixe.

Dans le B 6700, les premiers bits de chaque emplacement sont réservés pour indiquer la nature de l'objet représenté dans le reste de l'emplacement. Ils forment le préfixe de l'emplacement. Le préfixe peut indiquer, entre autres, une valeur primitive, un nom ou un descripteur.

L'utilisation systématique des préfixes a des incidences:

- sur l'adressage : quand le processeur accède à un emplacement que son préfixe signale comme étant un nom, alors que l'instruction nécessite une valeur primitive, il effectue automatiquement un accès supplémentaire à l'emplacement désigné par le nom. Ce chaînage automatique est répété jusqu'à la rencontre d'un préfixe signalant une valeur primitive.
- sur la fiabilité : quand le préfixe de la valeur n'est pas compatible avec le code opération de l'instruction qui y accède, une erreur est signalée. Lorsqu'un objet de mode repère ne repère encore aucune valeur, l'emplacement qui le représente contient une valeur indéterminée précisée par un préfixe.
- sur le nombre des codes opérations du langage machine : les opérations effectuées dépendent du type des opérandes. Ceux-ci peuvent parfois ne pas être identiques: par exemple, un opérande peut être en simple précision, l'autre en double précision. Le nombre des codes opérations s'en trouve diminué. Ainsi, il n'y a qu'un seul code pour tous les types d'addition.
- sur la compilation : les compilateurs sont simplifiés, car ils n'ont pas à engendrer des ordres d'indirection et de conversion de type.

6.22. Description des processeurs câblés.

Le langage machine est dérivé d'ALGOL 60. Nous employons le terme de texte-source pour tout programme écrit dans ce langage ou dans un autre langage de programmation (FORTRAN, COBOL). Une installation peut comporter plusieurs processeurs câblés. Chacun d'eux exécute des objets-procédures dont les instructions sont décrites en notation postfixée et gère en pile son espace de travail. Cette technique permet de traiter simplement la structure de bloc et les appels de procédure récursifs. En outre, de nombreuses instructions trouvent leur opérande sur le sommet de la pile qui n'a pas alors à être désigné.

La pile d'un processeur est contenue dans un segment, dont le descripteur est réparti entre deux registres: le registre base de pile qui contient l'adresse de base du segment et le registre plafond de pile qui contient

l'adresse du dernier mot du segment. Ces deux registres délimitent le volume total réservé pour la pile. A tout instant, l'adresse du sommet de la pile est contenue dans un registre sommet de pile, mis à jour par câblage à chaque opération d'empilement ou de dépilement. Le plafond détermine la position limite du sommet: une tentative de dépassement provoque un déroulement, qui entraîne le réajustement du plafond ou la destruction du processus. Il existe 16 registres notés *R0*, *R1*, ... *R15* dont nous verrons l'usage ultérieurement.

Les instructions sont rangées dans des segments-procédures, suites d'emplacements portant chacun le préfixe inst.

L'instruction qu'un processeur est en train d'exécuter est désignée par le registre pointeur d'instruction. Il contient le nom de l'emplacement où se trouve le descripteur du segment-procédure courant, et indique le déplacement de l'instruction courante, mesuré en octets par rapport à l'origine du segment-procédure courant.

Les interruptions sont traitées comme des appels de procédures inattendus, forcés par câblage. La sauvegarde du contexte au moment de l'interruption, comme sa restauration une fois l'interruption prise en compte, sont réglées par le schéma d'appel et de retour de procédure.

6.23. Les segments.

La mémoire physique se compose d'une mémoire centrale et d'un disque. Elle est allouée par zones à des segments, qui peuvent être de deux types:

- les segments de données, auxquels nous associons les piles;
- les segments-procédures qui contiennent des instructions exécutables.

Un segment est représenté par un descripteur de segment qui occupe un emplacement. Ce descripteur contient, entre autres, les informations suivantes:

- un préfixe (sd pour un segment de données, sp pour un segment-procédure),
- l'adresse de la base du segment en mémoire physique,
- la taille du segment.

La donnée du couple (*descripteur de segment*, *déplacement*) permet à un processeur de calculer une adresse en mémoire physique par addition du déplacement à l'adresse de base. Au cours de ce calcul d'adresse, le processeur vérifie que le déplacement n'est pas supérieur à la taille du segment et qu'il n'y a pas violation de protection. Nous ne considérons pas, dans ce chapitre, les déplacements des segments en mémoire physique et les mises à jour de descripteurs qui en résultent.

6.3. REPRESENTATION DES OBJETS.

Nous considérons ici un processus isolé disposant d'une pile. Tous les objets accessibles au processus sont représentés dans cette pile. Nous nous intéressons aux objets définis par le langage ALGOL 60.

6.31. Repères et valeurs élémentaires.

Nous supposons, pour simplifier, qu'il n'existe qu'un seul type de valeurs élémentaires, représentées sur un seul emplacement et préfixées par val. Un repère est représenté par un emplacement qui contient la valeur repérée, préfixée par val, ou une valeur indéterminée, préfixée par ind.

6.32. Tableaux.

Un tableau à une dimension (vecteur) est représenté sur un segment indépendant de la pile. Dans la pile, c'est le descripteur de ce segment qui représente donc un tableau.

Un tableau à deux dimensions de taille (l, c) est considéré comme un vecteur de l vecteurs de c éléments. Ses composants sont donc représentés sur l segments, dont les descripteurs sont regroupés dans un segment dont le descripteur représente le tableau. C'est ce descripteur qui représente le tableau dans la pile.

Remarques 1) On peut représenter, par cette méthode, un objet composé d'ensembles de taille variable.

2) Dans le B 6700, on peut représenter dans la pile un élément de tableau (c'est un repère), sous la forme d'un descripteur particulier constitué: - d'un préfixe el,
- de l'adresse de base du segment,
- d'un indice dans le segment.

Une telle représentation est intéressante si, dans le langage, on peut établir par exemple une relation de désignation entre un identificateur (paramètre formel par exemple) et un repère fourni par $T[I]$, entre un identificateur et un vecteur fourni par $\{T[I, i] / i = 1 \text{ à } n\}, \dots$, ce qui n'est pas le cas d'ALGOL 60.

6.33. Procédures, blocs et descriptif.

Chaque procédure est représentée dans un segment-procédure. Tout bloc, excepté s'il délimite le corps d'une procédure, est assimilé à une notation de procédure sans paramètre et à ce titre est représenté dans un segment-procédure.

Pour des raisons que nous expliciterons ultérieurement, l'ensemble des descripteurs de segment-procédure, qui définissent le programme utilisé par un processus, est conservé dans un descriptif, segment particulier, dont le descripteur est dans le registre *R1*. Chaque segment-procédure a un nom, qui est le numéro de l'emplacement du descriptif qui contient son descripteur. Ce nom est alloué statiquement. Le descriptif est de taille fixe pour chaque processus, et est constitué avant que le processus soit activé.

Nous verrons ci-dessous comment on représente une procédure dans la pile.

6.4. ACCES AUX OBJETS.

6.41. Objets manipulés par une procédure.

- 1) Une procédure est définie dans un contexte particulier qui contient outre les objets primitifs certains objets dits globaux; elle désigne ces deux types d'objets par des identificateurs que nous appelons globaux. Au moment de l'exécution de la procédure, la relation de désignation entre ces identificateurs et ces objets est établie.
- 2) A l'appel d'une procédure, il faut établir par contre des relations de désignation entre ses identificateurs de paramètre formel et des objets appartenant à l'état d'exécution de la procédure appelante.
- 3) Lorsque la procédure est appelée, elle établit des relations de désignation entre ses identificateurs locaux et des objets qu'elle crée pour toute la durée de vie de son exécution.
- 4) Lorsque la procédure s'exécute, elle établit des relations de désignation temporaires entre des expressions (réduites parfois à l'occurrence d'un identificateur ou d'une notation) et des objets de son état d'exécution.

6.42. La zone d'évaluation.

Pour évaluer une expression, le processeur manipule des noms, des valeurs ou des descripteurs dans une zone, appelée zone d'évaluation, gérée en pile. Seul l'emplacement en tête de zone doit être désigné: il l'est par le contenu du registre sommet de pile (fig. 1).

6.43. La région des locaux.

Considérons l'établissement des relations de désignation entre un identificateur local à une procédure et un objet, à durée de vie égale à celle de l'exécution de la procédure. On appelle région des locaux l'ensemble des emplacements

nécessaires pour représenter ces objets. De la même manière que dans le système CLICS, chacun de ses emplacements est désigné par un couple (reg, i) où reg est le nom d'un registre dont le contenu désigne la base de la région et i est un numéro d'emplacement.

L'exécution d'une procédure commence par créer, en tête de pile, une région représentant ses objets locaux.

Le couple (reg, i) permet au processeur d'obtenir le nom de l'emplacement, grâce au contenu de reg auquel il ajoute i . (fig.1, avec $reg = R3$)

6.44. Etat d'exécution à l'appel d'une procédure.

L'état d'exécution à l'appel d'une procédure $\mathcal{P}$ résulte d'exécutions successives, imbriquées ou récursives d'autres procédures. Supposons que ces procédures ne désignent que des objets locaux. Par le jeu des appels imbriqués (éventuellement récursifs) des procédures, on a créé des ensembles d'emplacements (région de locaux, zone d'évaluation - éventuellement vide) dont les durées de vie sont imbriquées. C'est pourquoi ils sont rangés au fur et à mesure de leur création dans une pile.

Ainsi, à l'appel de $\mathcal{P}$, et compte tenu de nos hypothèses, celle-ci trouve un état d'exécution représenté par l'ensemble des régions de locaux créées successivement dans la pile. Parmi cet ensemble, il y a des objets que $\mathcal{P}$ peut désigner par un identificateur global à $\mathcal{P}$, mais local à une autre procédure, d'autres qu'elle peut désigner par un identificateur de paramètre formel.

6.45. La région des liens formel/effectif.

Comme dans le système CLICS, c'est la procédure appelante qui connaît l'objet désigné ou fourni par un paramètre effectif. Chacun de ces objets est représentable sur un emplacement. Aussi, elle crée une région de taille égale au nombre de paramètres. Chaque emplacement de cette région est un lien contenant un exemplaire de la valeur de l'objet désigné.

La procédure appelée peut, dès son entrée, effectuer certaines opérations sur ces liens. En particulier, elle peut substituer à une valeur de type repère la valeur repérée. (L'objet est considéré alors comme un objet local, et le lien devient la représentation d'un repère.) Ces opérations effectuées, elle crée après cette région la région des locaux. Elle désigne chaque emplacement de l'une de ces régions grâce au couple (reg, i) décrit en 3.43. Le contenu du registre reg désigne donc la base des deux régions, lorsque la procédure a des paramètres.

6.46. Le contexte d'une procédure.

Représentons un programme source comme une arborescence $\mathcal{A}$ dans laquelle la racine est associée à la procédure constituant le programme, chaque noeud à une procédure; chaque arc représente la relation d'inclusion entre deux procédures. On appelle niveau d'une procédure le nombre $i + 2$, où i représente la longueur du chemin conduisant de la racine au noeud associé à cette procédure.

On appelle lexique local d'une procédure l'ensemble des identificateurs qui sont déclarés dans cette procédure, non compris ceux qui seraient déclarés dans une procédure interne.

On appelle lexique formel d'une procédure l'ensemble de ses identificateurs de paramètre formel.

Une procédure $\mathcal{P}$ de niveau K peut utiliser tous les identificateurs des lexiques locaux et formels des procédures l'englobant: ces procédures sont associées dans $\mathcal{A}$ aux noeuds conduisant au noeud associé à $\mathcal{P}$. Leurs niveaux respectifs sont $2, 3, \dots, K-1$.

La relation de désignation entre ces identificateurs et les objets qu'ils désignent est établie au moment où la procédure est appelée: ces objets sont tous désignés par un couple (reg, i) . Appelons C_K le contexte formé des objets susceptibles d'être désignés par une procédure de niveau K .

C_K est un ensemble de couples

(I , objet désigné)

représentés par des couples

((reg , i) , nom de l'objet)

Considérons les registres $R_2, R_3, R_4 \dots$. Soit, pour chaque procédure de niveau j , une numérotation $\mathcal{N} = \{2, 3, 4 \dots\}$ associée à la concaténation de son lexique formel et de son lexique local. Chaque identificateur de ces lexiques est représenté par le couple

(R_j , i) , $i \in \mathcal{N}$

et le contexte C_K se représente ainsi:

$$C_K = \bigcup_{j=2}^{K-1} \bigcup_i \{ (R_j, i) , \text{nom déduit du contenu de } R_j \text{ et de } i \}$$

Remarque: Les emplacements 0 et 1 de la région désignée par $\{(R_j, i)\}$ sont réservés à une zone dite de liaison. (fig. 1)

Le nombre d'emplacements désignés par les couples $\{(R_j, i) , i \text{ variant}\}$ est connu dès la compilation. On en déduit qu'on peut représenter le contexte C_K par l'ensemble:

{ (R_j , contenu de R_j) / $j = 2, \dots K-1$ }

Les noms des registres R_j se déduisant du niveau K , on peut réduire cette représentation au couple:

($K-1$, {contenu de R_j / $j = 2, \dots K-1$ })

. Le contexte particulier d'une procédure de niveau K (contexte des objets globaux) est par définition un sous-ensemble de C_K . Dans B 6700, on ne considère pas le contexte particulier à la procédure, mais le contexte C_K . Ce faisant, on évite la création de liens associés à la procédure pour accéder aux seuls objets globaux qu'elle désigne. En effet, le lexique de son contexte est supposé être par définition le lexique de tous les identificateurs locaux et formels des procédures englobantes. On dit que le contexte (particulier) de la procédure est celui des objets désignables (et non désignés effectivement).

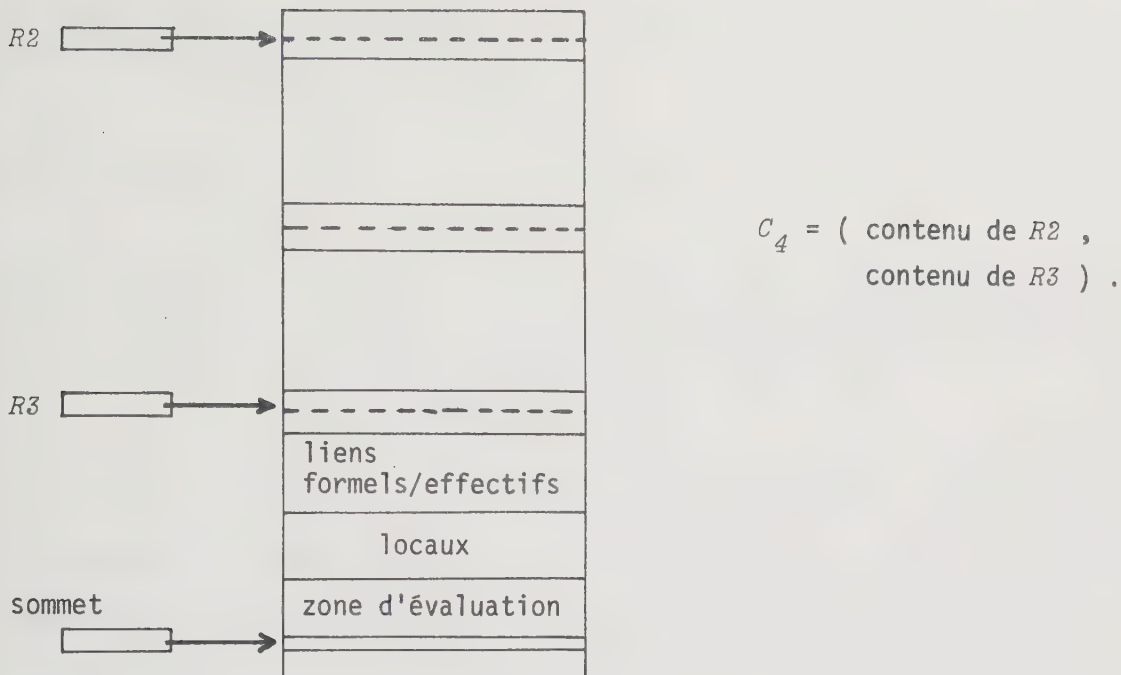


Fig. 1. Etat d'une pile pendant l'élaboration d'une procédure de niveau 3.

6.47. Représentation d'une valeur de type procédure.

Chaque fois qu'on crée une valeur de type procédure, on doit créer un objet composé de ses instructions et de la représentation de son contexte. Le lexique associé au contexte étant immuable, il n'est pas représenté. Par contre, comme dans le système CLICS, on ne connaît pas à priori les objets désignés dans la pile. Une solution consiste à associer à chaque segment-procédure un segment de liaison contenant C_K , et ceci à chaque élaboration de l'objet-procédure (c'est en effet à chaque déclaration de la procédure qu'il est nécessaire de garnir ce segment, car la procédure est élaborée à chaque fois dans un contexte différent).

Ce modèle de représentation peut être réduit. En premier lieu, on observe que lorsqu'une procédure $\mathcal{P}_{K-1}$, de niveau $K-1$ élabore plusieurs procédures (de niveau K), celles-ci ont le même contexte C_K . Ainsi, le segment de liaison peut être associé, non à la procédure $\mathcal{P}_K$ mais à $\mathcal{P}_{K-1}$. En second lieu, la

procédure $\mathcal{P}_{K-1}$ a été élaborée par une procédure de niveau $K-2$ à laquelle on a associé un contexte C_{K-1} , tel que

$$C_{K-1} \wedge C_K = C_{K-1}$$

Ainsi, C_K peut être réduit à la seule donnée d'un triplet:

$$C_K = (K-1 , \text{contenu de } R_{K-1} , \text{chaînage sur } C_{K-1})$$

C'est ce qui est fait dans la B 6700. A l'appel de la procédure $\mathcal{P}_{K-1}$, on crée une zone de liaison destinée à contenir C_K . Comme cette zone de liaison est désignée par le contenu de R_{K-1} , on peut encore réduire C_K :

$$C_K = (K-1 , \text{chaînage sur } C_{K-1})$$

Le chaînage sur C_{K-1} s'appelle lien statique ([2], [3]).

Il résulte de ces réductions que la représentation d'une valeur de type procédure est la suivante:

- la désignation (R_1, p) du descripteur de segment-procédure (connue à la compilation),
- l'indice i , dans le segment-procédure, de la première instruction (connue à la compilation),
- le nom de la zone de liaison contenant C_K (calculable à l'exécution).

Dans un objet-procédure, on peut désigner une telle procédure par le triplet

$$((R_1, p) , i , R_{K-1})$$

puisque R_{K-1} désigne à l'exécution la zone de liaison contenant C_K . Dans le B 6700, afin d'homogénéiser les désignations d'objets, on la désigne par le couple:

$$(R_{K-1}, j)$$

j étant le rang d'un emplacement dans la région des locaux de la procédure $\mathcal{P}_{K-1}$. Cet emplacement, préfixé proc, renferme le couple constant

$$((R_1, p) , i)$$

Lorsque la procédure est passée en paramètre, le lien qui la désigne doit contenir le triplet

$$((R_1, p) , i , \text{nom de la zone de liaison contenant } C_K)$$

Dans B 6700, ce lien contient le couple

$$(\text{nom de la zone de liaison contenant } C_K , j)$$

Afin d'homogénéiser le chaînage par lien, tout paramètre effectif est désigné par un tel couple. Un nom est représenté dans la pile par un couple (b, j) , b désignant une zone de liaison, j un déplacement dans l'une des régions créées à la suite de cette zone.

Remarque: Pour la description de l'appel et du retour de procédure, qui mettent en oeuvre des mises à jour des registres R_j , des créations et des destructions de zones de liaison, on peut consulter [2], [3]. Ces mécanismes nécessitent la conservation dans chaque zone de liaison d'un lien dynamique (chaînage inverse de toutes les zones de liaison créées) et de l'adresse de retour dans la procédure appelante.

6.5. PARTAGE DES OBJETS ENTRE UN PROCESSUS PERE ET SES PROCESSUS FILS.

6.51. Création de processus.

La création de processus se fait par une instruction d'activation sur une procédure.

Exemple: début entier B ;
 processus $TRUC$;
 procédure $P(I)$; entier I ; début ... fin ;
 début entier M ;

 créer activer $TRUC$ sur $P(M)$
 fin
 fin

On utilise la terminologie "processus père" et "processus fils" pour évoquer le processus créateur et le processus créé.

Toutes les possibilités des appels de procédures sont permises lors de la création d'un processus sur une procédure: les paramètres sont autorisés, la procédure peut avoir été déclarée à n'importe quel niveau, elle peut être elle-même un paramètre.

A l'exécution, une création de processus diffère cependant d'un appel de procédure sur les points suivants.

- Les régions créées par le processus fils sont rangées non pas au sommet de la pile du père, mais dans une pile nouvelle, propre au fils.
- Un nouveau processeur virtuel est mis en activité. Ses registres reçoivent l'information permettant de gérer le nouveau processus. Ce processus est lancé, et le processus père, dont les registres n'ont pas été modifiés, reprend immédiatement sa tâche.

6.52. Contexte commun à deux processus.

Soit $\mathcal{P}_i$ le processus père, et $\mathcal{P}_j$ le processus fils créé sur la procédure $\mathcal{P}$. Celle-ci, de niveau n , est déclarée dans la procédure $\mathcal{P}$ de niveau $n-1$. Le contexte de la procédure $\mathcal{P}$ est représenté par les contenus des registres R_j , $j=2, \dots, n-1$. L'ensemble des objets désignés dans ce contexte sont donc partagés par les processus $\mathcal{P}_i$ et $\mathcal{P}_j$. Il en est de même des objets désignés éventuellement par les paramètres de $\mathcal{P}$.

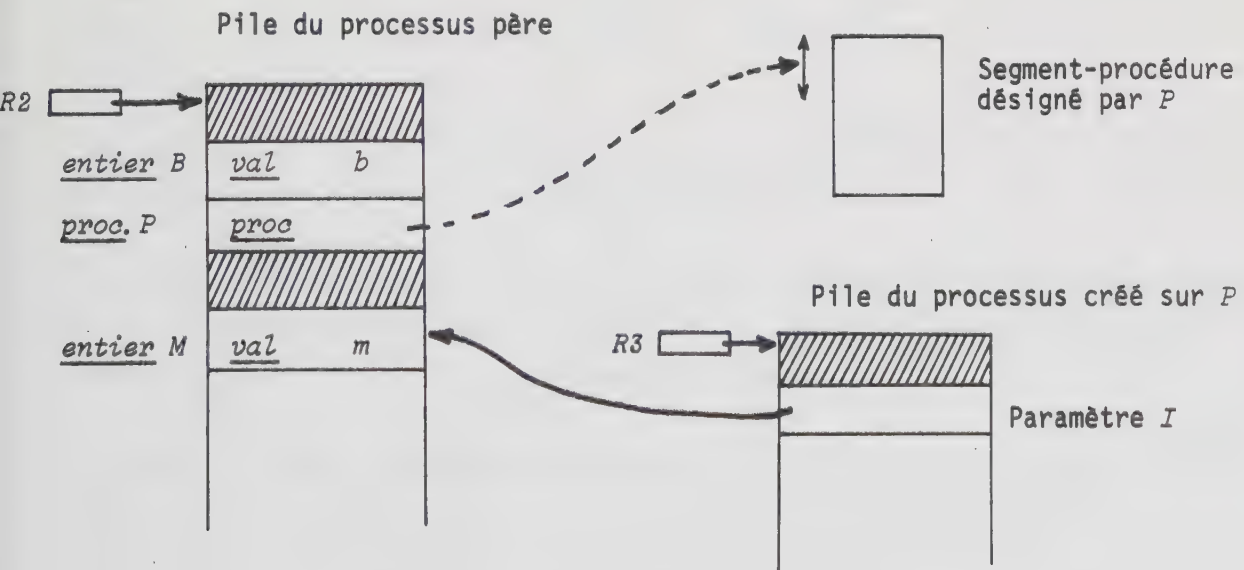


Fig. 2. Piles d'un processus père et d'un processus fils.
(voir exemple en 3.51.)

- À l'exécution, une création de processus diffère cependant d'un appel de procédure sur les points suivants.
- Les régions créées par le processus fils sont rangées non pas au sommet de la pile du père, mais dans une pile nouvelle, propre au fils.
 - Un nouveau processeur virtuel est mis en activité. Ses registres reçoivent l'information permettant de gérer le nouveau processus. Ce processus est lancé, et le processus père, dont les registres n'ont pas été modifiés, reprend immédiatement sa tâche.

3.5.2. Contexte commun à deux processus.

Soit P_1 le processus père, et P_2 le processus fils créé par la procédure P_1 . Celle-ci, de niveau n , est déclenchée dans la procédure P_2 de niveau $n-1$. Le contexte de la procédure P_1 est représenté par les contours des registres $R_1, R_2, \dots, R_n$. L'ensemble des objets désignés dans ce contexte sont donc partagés par les processus P_1 et P_2 . Il en est de même des objets désignés éventuellement par les paramètres de P_1 .

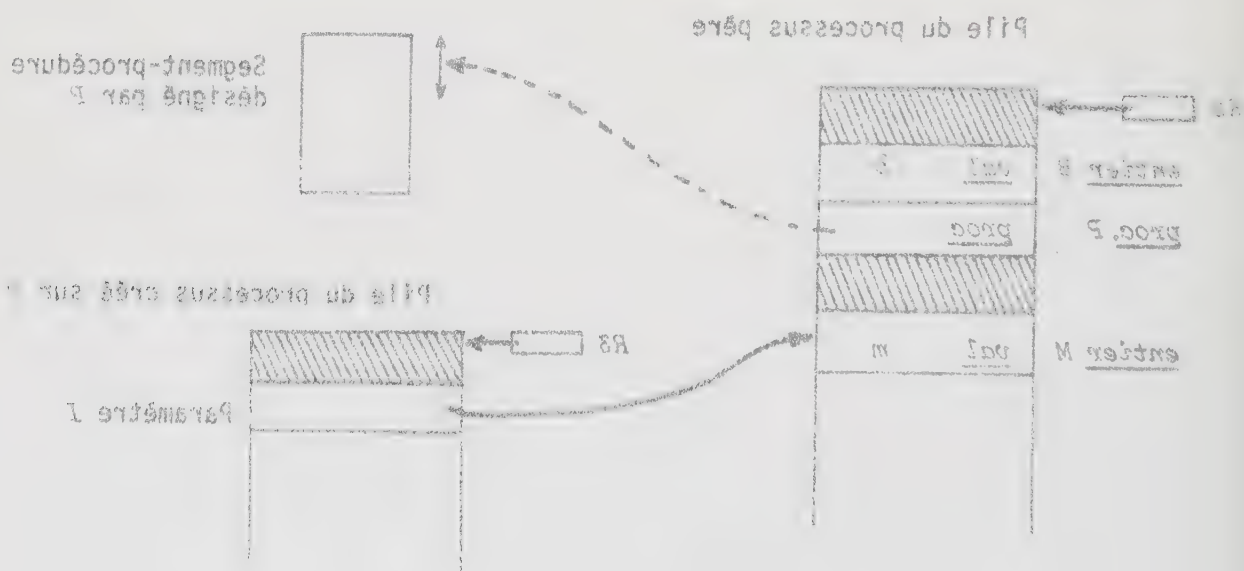


Fig. 2. Piles d'un processus père et d'un processus fils.
(voir exemple en 3.5.1.)

Dans la fig. 2, les objets accessibles à $\mathcal{P}$ sont:

- les objets de son contexte désignés par B et P (c'est-à-dire elle même),
- l'objet désigné par M , qui fait partie de son contexte de paramètre.

Le contexte mis en place dans le processus $\mathcal{P}_j$ évolue à partir du contexte de $\mathcal{P}$ selon les règles édictées en 6.4. La fin du processus $\mathcal{P}_j$ se produit lorsque la pile associée devient vide.

Un processus fils peut créer à son tour des processus fils. Les piles ainsi créées constituent une arborescence.

6.53. Désignation des objets.

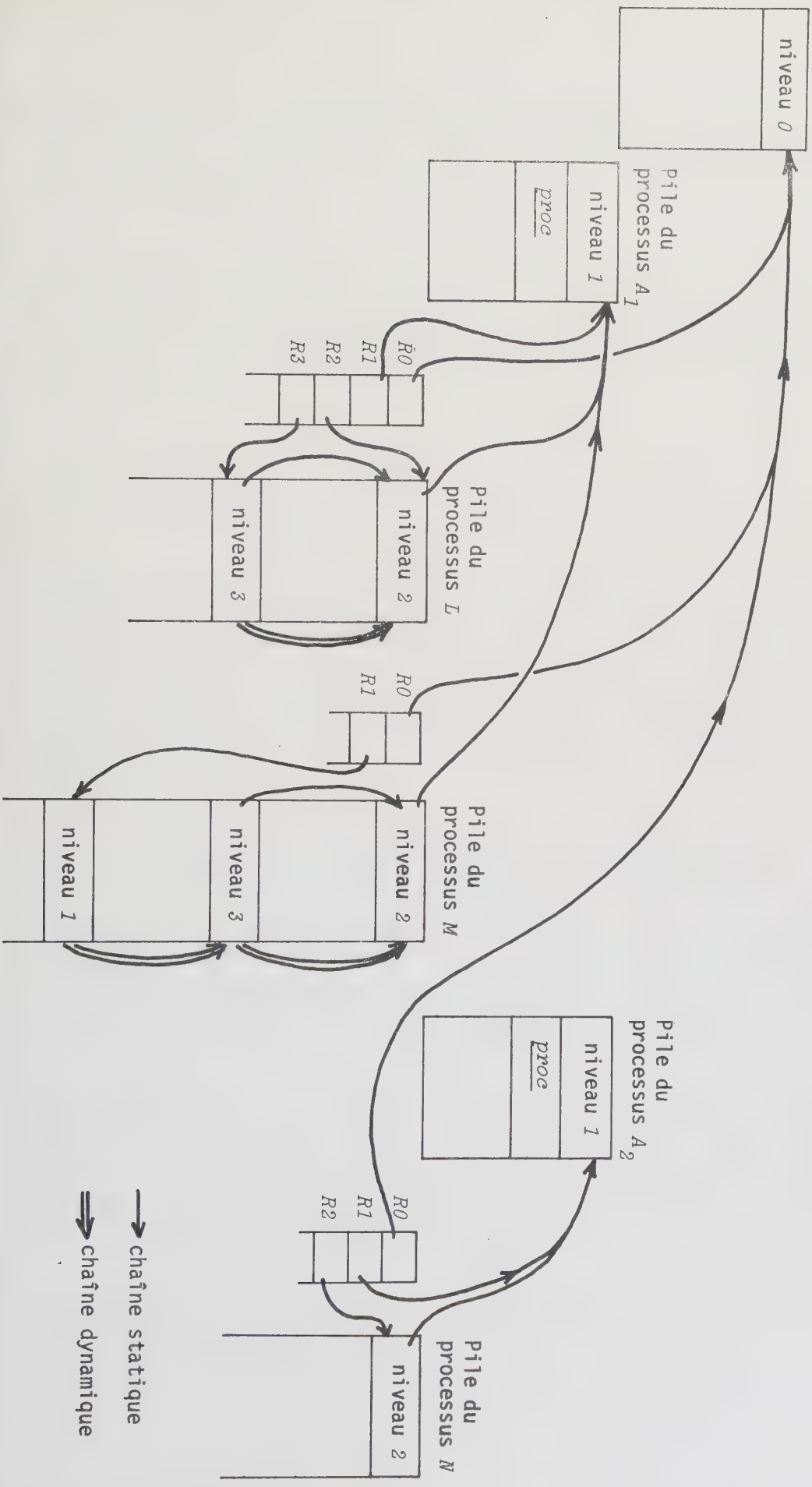
L'état d'exécution d'un processus $\mathcal{P}_i$ est caractérisé par les objets suivants:

- la pile sur laquelle il travaille,
- l'ensemble des régions appelé tronçon qui existaient dans la pile de son père au moment où il a été créé. Les emplacements de ces régions sont tous potentiellement désignables par un identificateur global ou un paramètre. Si son père et ses ancêtres sont des processus, il faut y ajouter des tronçons analogues. L'espace adressable de $\mathcal{P}_i$ est sa propre pile et un ensemble de tronçons des piles ancêtres.
- le contenu de ses registres $R2, R3, \dots$ qui désignent des zones de liaison dans son espace adressable, et ceux de ses autres registres.

On déduit de ces propriétés que les liens statiques peuvent désigner des zones de liaison appartenant à des piles différentes. Aussi le lien est un couple (numéro de pile, indice d'un emplacement dans cette pile).

Par contre les liens dynamiques ne désignent jamais que des zones dans la pile d'un processus. Quand la procédure appelée par la création d'un processus se termine, il n'y a plus aucun retour à effectuer et le processus est détruit. Aussi, un lien dynamique est un indice d'un emplacement dans la pile qui le contient.

Pile du processus
racine



→ chaîne statique

⇒ chaîne dynamique

La représentation d'un identificateur dans un segment-procédure n'a pas à être modifiée. C'est un couple (R_j, i) interprété comme un nom grâce au contenu de R_j . Par contre, un nom (b, i) devient un triplet puisque b doit être représenté comme un lien statique.

Le nom d'une pile est un indice dans un segment particulier, le descriptif des piles, qui regroupe les descripteurs de toutes les piles. Le descripteur du descriptif est un objet primitif représenté dans un emplacement de nom connu (voir 6.54.).

6.54. Partage des objets programmes et des objets primitifs par une collection de processus.

Considérons un objet que deux processus $\mathcal{P}_1$ et $\mathcal{P}_2$ indépendants veulent se partager. Nous avons vu qu'un moyen de le faire est que $\mathcal{A}$ soit commun aux espaces adressables des deux processus. Dans B 6700, aucun autre moyen n'a été mis en oeuvre: un processus ne peut pas établir dynamiquement une relation de désignation entre un identificateur et un objet qui n'appartient pas à son espace adressable.

Ainsi, un programme, partageable par définition, et représenté par un descriptif (voir 6.3.), est dans l'espace adressable des processus créés sur lui. Le descriptif est en effet une région d'objets locaux d'un processus créé sur une procédure particulière de niveau 1. C'est ce processus qui crée des processus fils sur le programme, procédure de niveau 2 par construction. C'est pourquoi, dans une pile, le nom d'un segment-procédure est un couple $(R1, i)$.

Les objets primitifs du système (procédures standards, descriptif des piles) sont désignables par les processus utilisateurs. Ces objets sont rangés dans une région d'objets locaux d'un processus racine supposé créé à la création du système sur une procédure de niveau 0. Tous les processus sont ses descendants. On désigne ces objets par le couple $(R0, i)$.

La représentation du système B 6700 est donc une arborescence de piles sur lesquelles s'exécutent des processus (fig. 3.).

Remarques: 1° Il faut empêcher qu'un processus père ne détruise le tronçon de sa pile qui appartient à l'espace adressable d'un processus fils, avant que ce dernier ne soit terminé. Dans ce but, à chaque fermeture de procédure, une procédure du moniteur (objet primitif) est appelée et vérifie qu'il ne subsiste pas de processus qui ont été créés par cette procédure.

2° Toute interruption, ou incident, provoque l'appel d'une procédure du moniteur. L'interruption peut être prise en compte par un processeur en charge du processus $\mathcal{P}_i$, même si l'interruption est destinée au processus $\mathcal{P}_j$.

BIBLIOGRAPHIE

[1] CLEARY J.C.

Process Handling on BURROUGHS B/6500

4th Australian Computer Conference, Adelaïde (1969)

[2] CREECH B.

Architecture of the B/6500

Infotech Report 2 (1971)

[3] HAUCK E., DENT B.

BURROUGHS B/6500 - B/6700 Stack Mechanism

SJCC (1968)

[4] ORGANICK E., CLEARY J.C.

A Data Structure Model of the B/6700 Computer System

in Proc. of a Symposium on Data Structures in Programming Languages
SIGPLAN NOTICES (1971)

Deuxième Partie

CONTRIBUTIONS A LA DEFINITION DES OBJETS
ET DE LEURS COMPOSITIONS DANS UN SYSTEME DE PROGRAMMATION

Chapitre 7. DIAMAG. Opérations sur les Contextes

Chapitre 8. Nature et Composition des Objets conservés
dans le Système SAR

CHAPITRE 7. DIAMAG . OPERATIONS SUR LES CONTEXTES.

7.1. INTRODUCTION.

Le système DIAMAG, conçu et mis en oeuvre en 1967-1968 (réalisé sur un calculateur IBM 7044, auquel était couplé un ordinateur PDP-8) permettait un accès conversationnel à un sous-système ALGOL 60.

Il est en dehors des limites de cet ouvrage de présenter le système DIAMAG [1]. On se limite ici à mettre en valeur quelques idées nées de l'expérience acquise grâce à l'exploitation de deux concepts: la conservation et le partage de procédures ALGOL 60 et l'utilisation de contextes au stade de l'exécution.

7.2. CONSERVATION ET PARTAGE DE PROCEDURES.

7.21. Conservation et partage de procédures dans DIAMAG.

Dans ce système, on peut conserver, dans un fichier, une procédure ALGOL 60 dont tous les objets, autres que ceux que son exécution doit créer, sont désignés par des paramètres formels. En d'autres termes, il n'existe pas, dans de telles procédures, d'identificateurs globaux. Dans ces conditions, l'élaboration d'une procédure est possible dans n'importe quel contexte.

Pour qu'une telle procédure soit partageable, il suffit que le mode de désignation de ses paramètres, de ses objets locaux et de ses composants internes (instructions par exemple) répondent aux conditions suivantes:

1° On considère que le niveau de nomenclature d'une procédure partageable est par définition le niveau-programme (dans le B 6700, c'est le niveau 2; voir 6.46.). Ses locaux et ses paramètres sont alors désignés par un couple (R_2, i) , les locaux et paramètres des procédures qu'elle élabore sont désignés par un couple (R_j, i) , $j = 3, 4, \dots$ ($R_2, R_3, \dots$ sont les noms des registres désignant les zones de liaison de chaque niveau de nomenclature; voir chap. 6.).

2° En ce qui concerne les composants de la procédure, ils sont désignés par un indice dans le segment-procédure, et accompagnés du niveau j de nomenclature de leur déclaration.

Ce faisant, nous mettons en évidence le fait que le contexte particulier d'une procédure n'est pas lié au niveau de nomenclature dans lequel sa valeur est élaborée.

7.22. Généralisation.

Dans DIAMAG, on note un défaut dans la conception du partage d'une procédure: c'est celui qui consiste à rendre obligatoire la désignation d'un objet non local par l'intermédiaire d'un paramètre. Les conséquences en sont les suivantes:

1° Chaque accès à un objet global nécessite un chaînage.

2° A chaque appel, on rétablit la relation de désignation avec l'objet désigné.

On peut éviter l'inconvénient 2° si on dispose d'un moyen d'établir une relation de désignation avec un objet global, à l'élaboration d'une valeur de type procédure (et non lors de ses appels), et pour toute la durée de vie de cette valeur. Une telle relation de désignation est alors établie dynamiquement. Si on n'évite pas l'inconvénient 1°, on gagne cependant un degré de liberté puisqu'à chaque élaboration d'une valeur de type procédure, on a une certaine latitude dans le choix de l'objet global désigné. Dans le modèle traditionnel, à chaque élaboration de valeur de type procédure, son contexte particulier contient éventuellement de nouveaux objets, mais il est toujours construit à partir d'un même lexique. Dans notre cas, le contexte effectif doit seulement être conforme à un modèle formel de contexte; par là, on impose simplement à chaque objet du contexte désigné d'avoir un type conforme à l'objet correspondant du contexte formel (voir exemple ci-dessous).

7.221. Notation d'une procédure.

Une procédure se présente sous la forme:

- d'une liste d'identificateurs globaux, formels, et de leur type,
- d'une liste de paramètres formels et de leur type,
- d'un corps de procédure.

Exemple (1) Un exemple d'écriture possible est le suivant:

contexte (entier I) par (réel J) :
< corps de la procédure >

7.222. Compilation indépendante d'une procédure.

Soit $\mathcal{F}$ le segment destiné à représenter un objet-procédure. A la compilation d'une procédure, on effectue les substitutions suivantes:

- 1) à chaque identificateur formel d'un objet global (liste contexte), on substitue un couple (C, i) . C est le nom d'un registre, i est le rang de l'identificateur dans le lexique. C est destiné à contenir le descripteur de la région des "liens aux objets globaux".
- 2) à chaque identificateur de paramètre (liste par), on substitue un couple (R_2, i) . Aux identificateurs locaux, on substitue un couple (R_j, i) , $j = 2, 3, \dots$ (voir chap. 6.).
- 3) à chaque étiquette locale, on substitue un couple formé d'un indice i dans le segment-procédure et d'un entier j précisant le niveau de nomenclature de sa déclaration ($j = 2, 3, \dots$).

La représentation de la procédure contient l'objet-procédure et un modèle de types des objets globaux (pour les paramètres, voir chap. 6.).

7.223. Elaboration, mise en relation et exécution d'une procédure.

Lorsque, dans un processus, on veut utiliser une telle procédure, il est nécessaire

- de faire intervenir une procédure primitive pour effectuer la recherche de l'objet dans un catalogue et vérifier les droits d'utilisation d'une telle procédure par ce processus: c'est la phase d'accès,
- de faire élaborer ensuite la valeur de la procédure. Dans ce but, on doit désigner les objets globaux choisis pour cette procédure. L'élaboration consiste à vérifier la conformité des types et à garnir la région des liens aux globaux, créée à cet effet: c'est la phase d'élaboration.

- de mettre en relation la valeur ainsi obtenue dans une chaîne d'accès; en ALGOL 60, on ne peut que la faire désigner par un identificateur. C'est la phase de mise en relation.

La procédure peut alors être exécutée.

Remarque: A des fins de fiabilité, il serait souhaitable que la mise en relation effectue une conformité entre un modèle formel d'en-tête de procédure et la valeur élaborée (semblable à celle qui est faite lorsqu'on effectue une désignation entre un identificateur de paramètre formel et un objet).

Exemple: Soit "LIP" la chaîne de caractères identifiant le segment où est représentée la procédure de l'exemple (1).

- a) exemple dans lequel la conformité n'est pas réalisée à la mise en relation:

procédure $P = \text{LIER ("LIP")} \text{ avec } (A)$

La procédure *LIER* délivre un objet procédure après avoir vérifié les droits d'utilisation. L'opérateur avec élabore la valeur de type procédure. L'opérateur = effectue la relation de désignation.

- b) exemple dans lequel la conformité est réalisée lors de la mise en relation:

procédure $P \text{ context } (\text{entier}) \text{ par } (\text{réel}) = \text{LIER ("LIP")} \text{ avec } (A)$

Dans ces exemples, *A* désigne un objet connu du processus. Après l'opération, *P* désigne la valeur de type procédure, équivalente en ALGOL 60, à celle qui est élaborée à la déclaration suivante:

procédure $P (J) ; \text{réel } J ;$

< corps de la procédure dans lequel on a substitué à toute occurrence de *I* l'identificateur *A* >

- c) les appels de procédure suivants ont un sens

$P (B)$

$Q (P)$ si *Q* est une procédure à paramètre de type procédure

$R (\text{LIER ("LIP")} \text{ avec } (U))$ si *R* est une procédure à paramètre de type procédure.

7.224. Mise en oeuvre.

Dans le cadre d'un ordinateur comme le B 6700, la mise en oeuvre est simple. Une valeur de type procédure peut être représentée par le triplet suivant:

- le nom du segment-procédure, partageable,
- l'indice, dans ce segment, de la première instruction,
- le descripteur d'un segment contenant la région des liens aux globaux. Ce segment est propre au processus.

Cette représentation a une taille fixe. Dans le B 6700, elle nécessiterait deux emplacements.

7.23. Conclusion.

Un tel mode de partage de procédures peut s'étendre à d'autres langages. Dans le cadre du système SAR, basé sur le langage ALGOL 68 (voir chap. 8.), nous avons cherché avant tout à exprimer la nature des objets et de leur utilisation en termes d'ALGOL 68. Dans ce langage, l'exécution d'une procédure peut rendre en résultat une autre procédure. Le mécanisme d'établissement du contexte particulier d'une procédure étant analogue à celui de l'établissement du contexte formé des paramètres, il était tentant d'exprimer la phase d'élaboration par une exécution de procédure.

Exemple: a) définition de l'objet

$$\begin{aligned} &\underline{proc} \ Q = (\underline{ent} \ I) \ \underline{proc} \ (\underline{réel}) : \\ &\quad (\ \underline{proc} \ P = (\underline{réel} \ J) : < \text{corps de la procédure} > \\ &\quad \quad P \\ &\quad) \end{aligned}$$

b) élaboration de la valeur et mise en relation

$$\underline{proc} \ P = Q \ (A)$$

Q délivre une valeur de mode procédure à paramètre réel. Malheureusement, la portée de P est limitée par celle de l'identificateur I . Aussi une telle écriture est incorrecte.

Dans SAR, c'est grâce à un nouveau concept, introduit en ALGOL 68, et appelé prolator [2], que nous pouvons pallier à cette difficulté.

7.3. UTILISATION DES CONTEXTES A L'EXECUTION. INTERACTIVITE.

7.31. Utilisation des contextes dans DIAMAG.

Dans DIAMAG [1], on peut suspendre l'exécution d'un programme ALGOL 60 en un point et faire élaborer et exécuter, dans l'état d'exécution courant, une notation de procédure conçue à ce stade.

L'exécution d'un programme est toujours suspendue, à la fin d'une "phrase", à la suite d'une interruption déclenchée par l'utilisateur, à la rencontre d'un appel de procédure spéciale ou bien, parce que le mode d'exécution choisi déroule le programme phrase par phrase.

Chaque phrase est la plus petite entité syntaxique $\langle s \rangle$, terminée par un délimiteur, telle que

$$E : \langle s \rangle$$

où E n'est pas une étiquette utilisée dans le programme, soit syntaxiquement correct.

Les notations de procédure sont écrites dans un langage simple, sous-ensemble d'ALGOL 60, comprenant des opérations de consultation de valeurs, de modification de relations de repérage et de rupture de contrôle. L'exécution d'une telle notation ne modifie que l'état d'exécution courant.

Pour mettre en oeuvre cette possibilité, chaque objet-procédure (au sens large incluant les blocs; voir chap. 6.) est accompagné de son contexte local ou de son contexte formel. Le contexte dans lequel chaque notation de procédure est élaborée comprend tous les objets des régions reliées par la chaîne statique (voir chap. 6.). C'est dans les zones de liaison qui préfixent ces régions dans la pile que sont rangés les chaînages sur les contextes correspondants. Chaque élément du contexte contient un identificateur, le type de l'objet désigné et la représentation de l'identificateur. Il n'y a aucune difficulté, dans ces conditions, à élaborer puis exécuter la notation de procédure.

7.32. Généralisation.

Dans DIAMAG, c'est uniquement une question de mise en oeuvre qui limitait le langage dans lequel étaient écrites les notations de procédure. Si l'appel du compilateur, pendant l'exécution, avait été techniquement envisageable, rien ne s'opposait à ce que le langage de conversation soit étendu à ALGOL 60 dans son intégrité. C'est ce qui a été envisagé dans les systèmes incrémentiels conçus à Grenoble [3] (langages ALGOL 60, FORTRAN, PL/1) et dans les systèmes conversationnels NPL [4] (langage ALGOL 60) et POP [5] (langage POP).

Dans ces systèmes, on construit un programme phrase par phrase, chaque phrase étant exécutée au fur et à mesure. L'exécution se présente comme un couple

(*< contexte >* , *< état d'exécution >*)

sur lequel l'utilisateur peut agir. Les opérations possibles sont principalement:

- étendre le contexte et l'état d'exécution grâce à une déclaration;
- modifier l'état d'exécution grâce à l'exécution d'une notation de procédure;
- supprimer un élément du contexte.

Le seul contexte considéré est celui qui est associé au niveau de nomenclature le plus externe (niveau programme). Les opérations de manipulation du contexte sont effectuées par l'intermédiaire d'un langage de commande.

Dans le système SAR [2], nous avons généralisé ces possibilités.

BIBLIOGRAPHIE

[1] VERJUS J.P.

Etude et Réalisation d'un Système Algol Conversationnel
Thèse Doc-Ing Grenoble (1968)

[2] TRILLING L., VERJUS J.P.

Introduction dynamique de processeurs dans un environnement conversationnel
Article proposé pour publication (1973)

[3] GRIFFITHS M.

Systèmes pour la compilation incrémentielle de PL/1, ALGOL et FORTRAN
dans *Les Systèmes Conversationnels*, BOLLIET L. (Ed.)
DUNOD (1969)

[4] MORVEN W., MURRAY W.A.

A proposed Specification of a Conversationnal Computing System for N.P.L.
National Physical Laboratory (1969)

[5] BURSTALL R.M., POPPLESTONE R.J.

The POP-2 Reference Manual
Machine Intelligence 2, Oliver-Boyd (1968)

CHAPITRE 8. NATURE ET COMPOSITION DES OBJETS CONSERVES DANS LE SYSTEME SAR.

8.1. INTRODUCTION.

Le premier objectif poursuivi dans la conception du système SAR est de mettre en valeur la frontière existant entre un langage de programmation et un langage de commande. Dans les fonctions offertes par un langage de commande, nous avons distingué trois classes.

1°) Les fonctions de manipulation d'objets dont l'expression, dans un langage de programmation, est envisageable, et les fonctions programmées de communication avec le monde extérieur, dont il existe un cadre de définition dans la plupart des langages.

2°) Les fonctions de communication interactive avec un utilisateur ou tout objet dont le comportement est non prédictible (interruption, autre programme). Ces fonctions ne sont généralement pas envisagées dans les langages de programmation.

3°) Les fonctions d'un processeur spécial, souvent implicite, qui sert d'intermédiaire entre l'utilisateur et les processeurs auxquels celui-ci veut faire effectuer des actions.

Le système SAR définit un processeur primitif, le noyau. Il est formé du prologue standard d'ALGOL 68 et d'un prologue de programmathèque que nous cherchons à définir aussi élémentaire que possible.

Tout autre processeur défini dans le système (de façon standard ou par les utilisateurs) est écrit dans le langage défini par le noyau.

Les commandes nécessaires à l'utilisation du système (en particulier celles qui assurent les fonctions de la troisième catégorie) sont exprimées à l'aide d'une procédure du noyau. Celle-ci est l'interlocuteur de tout utilisateur. Toute action effectuée par un utilisateur est contrôlée par l'exécution de cette procédure.

Un utilisateur est représenté

- par les caractères enregistrés sur un terminal, de manière interactive,
- ou par les caractères pré-enregistrés sur un paquet de cartes, ou un quelconque dispositif périphérique,
- ou enfin par les caractères pré-enregistrés dans un fichier.

La définition du système résulte donc de l'expression

- d'un prologue de programmathèque,
- d'une procédure de contrôle,
- et de l'expression, dans le prologue de programmathèque, de l'activation d'autant d'exemplaires de la procédure de contrôle qu'il existe d'utilisateurs simultanés.

Le noyau du système SAR a donné lieu à une première version, destinée à un seul utilisateur [1], et à une mise en oeuvre rudimentaire [2]. Nous avons apporté à cette première version des modifications [3] et nous cherchons actuellement à préciser les mécanismes d'appel de la procédure de contrôle.

Nous ne traitons, dans ce chapitre, que de la nature et de la composition des objets conservés dans le système.

8.2. NATURE ET GESTION DES OBJETS DEFINIS DANS UN ENVIRONNEMENT ALGOL 68.

Nous avons vu que, dans CLICS (chap. 5.), comme dans MULTICS, les objets conservés sont liés dynamiquement à un processus, et que l'accès à leurs composants est analogue à l'accès aux objets propres du processus.

Dans le langage ALGOL 68 [4, 5], il existe un cadre précis de définition des échanges entre un programme et son environnement. Dans [6], nous l'avons comparé à différents systèmes de fichiers, et nous avons proposé, pour le compléter, un ensemble de procédures qu'il est souhaitable de définir dans un prologue de programmathèque.

Nous reprenons brièvement l'exposé des fonctions, relatives à la gestion et à la manipulation d'objets conservés, qui sont définies dans le prologue standard ou qu'il est envisageable de définir dans le prologue de programmathèque. Nous les comparons, au fur et à mesure, à celles qui sont offertes dans le système CLICS.

Nous appelons livres les objets soumis aux échanges en ALGOL 68. Chaque livre est composé:

- de son texte rangé sur p pages de l lignes de c caractères,
- de sa taille précisée par le triplet p, l, c ,
- de son identification, précisée par une chaîne de caractères.

Remarques: 1) Dans le langage [4, 5, 6], le texte, appelé livre, est représenté comme un tableau tridimensionnel d'entiers. Chaque entier est la codification d'un caractère qui, au moment d'un échange, est mis en correspondance avec la représentation d'un caractère ALGOL 68.

2) Dans le langage révisé [7], le texte est représenté comme un tableau de tableaux de tableaux de caractères.

3) Nous avons adopté ici la terminologie du langage révisé [7].

Nous appelons processus l'exécution d'un programme particulier ALGOL 68. En ce qui concerne les processus de CLICS, nous en conservons la définition donnée au chapitre 6.

8.21. Détermination des objets utilisables par un processus.

Supposons qu'un objet $\mathfrak{x}$ existe dans l'environnement (voir 8.24.), et qu'on connaisse un moyen de le désigner.

Dans CLICS, on vérifie, lors de la première désignation de $\mathfrak{x}$ par un processus $\mathfrak{p}$, que $\mathfrak{x}$ existe d'une part, et que le processus a le pouvoir de l'utiliser. L'objet est alors lié au processus par le biais du rangement de son descripteur dans le descriptif du processus.

En ALGOL 68; le langage précise que le livre, pour être accessible, a dû être mis en relation avec une valeur structurée, particulière au processus, et appelée *chaînevolume*. Ainsi, *chaînevolume* joue le rôle d'un descriptif. Il est très simple, dans ces conditions, de définir une procédure qui lie un livre au processus après en avoir vérifié l'existence et le droit du processus à l'utiliser. Nous appelons une telle procédure *mettre chaînevolume*. Nous en préciserons les caractéristiques en 8.3.

8.22. Mise en relation de l'objet et rupture de la relation.

Dans CLICS, l'éditeur de lien, appelé implicitement lors de la première désignation de l'objet $\mathfrak{X}$ par une procédure $\mathfrak{P}$ du processus, a automatiquement mis $\mathfrak{X}$ en relation avec $\mathfrak{P}$ (il en a fait un objet accessible à $\mathfrak{P}$ en établissant la relation de désignation). Toutes les autres procédures appelées dans le processus peuvent désormais désigner l'objet (voir cependant 8.23.).

Les relations établies sont généralement valables pour toute la durée du processus. Elles sont rompues si l'objet est détruit avant la fin du processus. L'accès à l'objet est alors oblitéré grâce à une marque de déroutement rangée dans son descripteur.

En ALGOL 68, c'est une opération d'ouverture qui donne accès à un livre. L'ouverture met en relation un livre rangé dans *chaînevolume* avec une valeur structurée de type fichier. Cette valeur structurée (nous dirons un fichier) précise, par certains de ses champs, les modalités d'utilisation de l'objet (voir 8.23.)

C'est une opération de fermeture qui rompt la relation entre le livre et un fichier. Le livre est alors relié à *chaînevolume*, et peut être mis de nouveau en relation avec ce même fichier ou un autre, par une nouvelle opération d'ouverture.

Ainsi, un fichier joue le rôle d'un lien, à cette différence près qu'il constitue un objet interne. Le lien est assuré par l'un de ses champs (accessible aux seules procédures *ouvrir* et *fermer*) qui est un repère de livre.

Le fichier a un second intérêt: il sert à exprimer la fonction d'accès au livre (voir 8.23.). On peut, par exemple, définir deux fichiers repérant le même livre: ils définissent simultanément deux fonctions d'accès.

L'intérêt des opérations d'ouverture et de fermeture s'en déduit. Par le jeu d'une fermeture suivie d'une ouverture, un même fichier peut servir de mode d'emploi à un livre puis à un autre. D'autre part, certains livres créés en mode séquentiel doivent être fermés puis réouverts pour être lus.

Dans CLICS, les fonctions d'accès à un objet catalogué ne sont pas définies à tout instant comme en ALGOL 68. C'est pourquoi les concepts de fichier, d'ouverture et de fermeture ne sont pas explicites (voir 8.23.).

8.23. Accès à l'objet.

Dans CLICS, l'objet, à sa création, est compilé (au sens large: nous appelons compilation l'opération qui convertit une notation d'objet en une représentation interne). L'objet compilé est représenté dans un segment, éventuellement structuré en une suite de régions. Ce découpage en régions est la seule structure retenue dans la représentation. Chaque région est la représentation d'un champ, dit de niveau 1, d'un objet composé. L'identificateur de champ est conservé dans le segment.

Exemple (1) Un objet composé (appelé data program) a pour notation une suite de déclarations.

```
declare 1 structure ,  
        2 tableau (9) integer ,  
        2 entier integer ;  
declare chaîne (8) character ;
```

Cet objet a pour premier champ une structure, *structure*, elle-même composée de deux champs. *tableau* est un vecteur de 9 repères d'entiers représentés grâce aux 9 premiers emplacements de la première région. *entier* est représenté sur le 10^e et dernier emplacement de cette région. Le deuxième champ de l'objet est une chaîne de caractères représentée à l'aide du seul emplacement de la seconde région.

Ces repères auraient pu repérer une valeur initiale.

Le segment est donc constitué de 2 régions précédées par des couples:

(x , "structure")

et ($x+10$, "chaîne")

représentés sur les x premiers emplacements du segment.

Un segment représente donc un contexte construit sur un lexique et dont chaque objet est représenté par une région du segment.

Lorsque l'objet a été lié à un processus, on a noté, dans le descripteur de sa représentation, un pouvoir d'utilisation maximum par ce processus (par exemple: accès seulement aux valeurs repérées). Un mécanisme, non décrit dans le chapitre 5., permet d'en limiter l'usage qu'en fait une procédure $\mathcal{P}$ pour le compte du processus, en-deçà du pouvoir d'utilisation. Ce mécanisme est celui des "anneaux de protection" [8, 9]. Le pouvoir d'accès à un objet x , de la part d'une procédure $\mathcal{P}$, exécutée par un processus Q , dépend:

- d'une part, du pouvoir d'utilisation de x par Q , calculé au moment de la liaison de x à Q ,
- et d'autre part, du pouvoir d'utilisation de x par $\mathcal{P}$, qui dépend des anneaux dans lesquels sont situés respectivement les objets x et $\mathcal{P}$, au moment de l'accès.

L'accès de $\mathcal{P}$ à x , s'il est possible, résulte des opérations suivantes:

1°) déclaration, dans $\mathcal{P}$, d'un type formel d'objet composé. Cette opération est semblable à l'affectation d'un format à l'un des champs d'un fichier ALGOL 68.

Exemple (2) *declare 1 struct based ,*
 2 table (8) integer ,
 2 un integer ,
 2 ~~deux~~ integer ;

2°) accès à un champ (ou un sous-champ) de l'objet composé, grâce à ce type formel.

Exemple: Si l'objet composé de l'exemple (1) a pour identification *BIBL.OBJET*
 "BIBL.OBJET.STRUCTURE" p → struct.un

fournit à la procédure $\mathcal{P}$ (voir exemple (2)) le nom du neuvième emplacement de la première région de *BIBL.OBJET*. Cet emplacement est la

représentation de l'objet désigné par
structure.tableau (9)
à la création de *BIBL.OBJET*.

L'introduction du mode formel (exemple (2)) permet au compilateur de générer de façon optimale l'accès à la région. "*BIBL.OBJET.STRUCTURE*" est converti par l'éditeur de lien, donc au stade de l'exécution, en l'adresse segmentée du premier emplacement de la région. Aussi, le compilateur doit seulement générer une opération d'incrémentation de cette adresse segmentée.

On note cependant qu'il ne s'opère aucune vérification de l'équivalence des types formels et effectifs, à l'exception de l'existence d'un champ d'identificateur *STRUCTURE* dans *BIBL.OBJET*.

En ALGOL 68, la fonction d'accès à un livre est précisée au moyen de trois concepts:

- a) le type d'accès au livre,
- b) le format associé au fichier,
- c) les procédures de contrôle.

Lorsque le livre est lié à un fichier, on précise un canal de liaison. Un canal (qui permet de rendre compte des caractéristiques d'un dispositif périphérique, ou qui permet de traduire des restrictions d'accès au livre) a des caractéristiques parmi lesquelles nous notons: le mode d'adressage (séquentiel, adressable), le genre d'opérations possibles (lecture, écriture), le type d'échange (valeur interne / texte "affichable", ou valeur interne / texte représenté sous forme d'une suite binaire).

Le format est une suite de caractères définissant une relation entre des valeurs internes et le texte du livre. Cette relation définit la syntaxe et la sémantique du texte lu ou à produire.

Chaque fichier est doté de champs qui peuvent repérer des procédures de contrôle définies par l'utilisateur. Ces procédures sont appelées lors d'incidents dans les échanges. C'est un aspect intéressant du langage; il permet par exemple de récupérer une erreur due à une lecture en dehors des bornes du livre.

Pour conclure sur ces méthodes d'accès nous notons qu'il faut se garder de comparer un format (ALGOL 68) et une déclaration formelle de type (CLICS). Le format contient évidemment une déclaration formelle de type, mais en outre des possibilités très variées d'édition (ou d' "indition") de représentation externe et d'interprétation du texte. Si on considère un format comme un traducteur paramétré de texte en objets internes ou vice-versa, son existence est alors indépendante des accès aux fichiers, conservés sous forme d'objets compilés. On gagne alors la rapidité dans les accès qu'offre l'utilisation d'un compilateur en lieu et place d'un interpréteur. On y perd évidemment en souplesse d'utilisation.

8.24. Création, destruction et conservation d'objets.

Dans le système CLICS, ce sont des procédures du moniteur qui créent et détruisent un objet. Tout objet créé est catalogué aussi longtemps qu'il n'est pas détruit.

En ALGOL 68, ce sont des procédures standards qui créent et détruisent des livres. Une procédure, *verrouiller*, a pour but d'indiquer que l'objet doit être confié à l'environnement. Son intérêt est principalement relié à la gestion de l'espace adressable puisque le verrouillage d'un livre par un processus lui en interdit tout accès ultérieur. Il faut évidemment compléter cette opération, en fonction du mode de conservation des objets. Nous étudions cet aspect au paragraphe suivant (8.3.).

8.25. Conclusion.

Nous avons constaté des différences conceptuelles dans la nature et l'utilisation des objets définis dans CLICS et en ALGOL 68. Souvent les avantages d'un choix dans un système sont les inconvénients du choix effectué dans l'autre système. Nous constatons cependant que CLICS offre l'avantage de pouvoir conserver des objets compilés de nature exécutoire ou non. Cette

possibilité est indispensable dans un système. C'est pourquoi, nous avons cherché à doter le système SAR de possibilités analogues, que nous présentons en 8.4.

8.3. LE CATALOGUE DES OBJETS CONSERVES.

L'étude des moyens à mettre en oeuvre pour conserver les objets dans le système nous a amenés à explorer deux voies.

Dans une première approche [1, 10], nous avons défini le catalogue comme un objet du prologue de bibliothèque, dont les composants étaient accessibles seulement au moyen de procédures.

Nous ne décrivons pas ici la structure alors définie. Elle est de conception classique et se présente comme une arborescence de livres. L'accès à chaque livre est contrôlé grâce à un mécanisme d'autorisation.

Le catalogue contient d'une part des livres, mais aussi des "boîtes aux lettres" [10] qui constituent un support de communication entre les utilisateurs.

Cette définition, qui est restée au stade d'un exercice formel, nous a permis de constater l'adéquation du langage à exprimer une telle structure d'objets. Elle nous a aussi permis de mettre en valeur quelques défauts.

Si le catalogue est un objet composé décrit dans le prologue de bibliothèque, il est commun à tous les utilisateurs. Ceci signifie que chacun de ses composants doit être rangé dans une zone commune à tous les processus du système, gérée en tas du fait du nombre variable de ses composants.

Par ailleurs, la définition du catalogue dans le prologue fige sa structure, ou tout au moins rend très difficile sa modification. Il peut être en effet très commode de vouloir modifier la structure du catalogue au vu de résultats expérimentaux.

Enfin, ce choix présentait un dernier inconvénient, d'ordre formel. Pour interdire aux utilisateurs un accès incontrôlé aux composants du catalogue, nous utilisons la marque % d'oblitération des identificateurs.

Exemple: Soit un repère d'entier inaccessible à l'utilisateur à l'aide de l'identificateur i . On écrit:

rep int % i = ...

Si on veut donner accès à la valeur repérée, on définit une procédure

proc val de i = int : i ;

Cette méthode de protection, facile à mettre en oeuvre, est en principe non sélective. Or, il doit exister dans le système un utilisateur privilégié, chargé par exemple d'enregistrer, dans le catalogue, le matricule d'un nouvel utilisateur. Pour compiler les "programmes" de cet utilisateur, il aurait fallu disposer d'un compilateur spécial autorisant l'emploi d'identificateurs tels que i dans l'exemple ci-dessus.

Afin de pallier à ces défauts, nous avons choisi de décentraliser la fonction "catalogue". Ce choix s'étend d'ailleurs à d'autres fonctions du système comme les échanges avec un lecteur de cartes, une imprimante, ...

Le système est, dans cette optique, à considérer comme un ensemble de processus parallèles. Certains de ces processus sont spécialisés et associés à un utilisateur fictif: le lecteur de cartes, l'imprimeur de papier, le bibliothécaire. Chaque processus a en propre un dispositif périphérique d'entrée et un autre (ce peut être le même) en sortie. En outre, chaque processus peut communiquer avec chaque autre. Ces communications respectent un protocole d'échange qui, seul, a à être défini dans le prologue de programmation. Ce protocole d'échange nécessite la mise en oeuvre:

- d'un objet composé, destiné à enregistrer des messages ou des livres. Il peut être de taille fixe, ce qui permet une gestion plus facile (voir 6.5.),
- de procédures de production et de réception de messages et de livres.

Dans cette optique, la procédure *mettre en chaîne le volume* (voir 8.21.) doit seulement envoyer un message au lecteur de cartes ou au bibliothécaire, et attendre sa réponse. La procédure *ouvrir* doit être synchronisée avec l'arrivée effective du livre. Quant à la procédure *verrouiller* (voir 8.24.), elle doit envoyer un message accompagné du livre concerné, à l'imprimeur ou au bibliothécaire.

8.4. NATURE, CONSERVATION ET UTILISATION D'OBJETS COMPILES.

La conservation d'objets compilés permet d'augmenter la rapidité des accès. Pour que l'utilisation de ces objets se fasse en toute sécurité, il est cependant nécessaire d'opérer une vérification de type. C'est ce que nous avons cherché à exprimer dans le cadre du langage. Nous exposons en 8.42. à quelles conditions les objets internes ALGOL 68 sont conservables.

Un second point a retenu notre attention: comment enrichir le système grâce à la définition de nouveaux types d'objets tels que les processeurs, modules ou fichiers présentés en 2.6., tout en restant le plus possible dans le cadre du langage. Cette condition était essentielle puisque le langage est considéré comme le seul support de communication et, qu'en outre, il oblige à une conception rigoureuse. Nous exposons en 8.41. un concept simple, le prolator, que nous avons ajouté au langage, et qui nous permet d'exprimer la nature de ces objets à l'aide de procédures.

Dans [11], nous montrons que le prolator est nécessaire parce que nous avons choisi d'offrir aux utilisateurs de SAR des possibilités de construction de processeurs en mode conversationnel (voir 7.32.). Nous résumons ci-dessous les fonctions de construction et d'utilisation, en mode interactif, d'un processeur, détaillées en [11].

Le langage de construction et d'utilisation d'un processeur est construit sur le vocabulaire

{ { , } , # }

en supposant qu'aucun de ces trois symboles ne peut figurer dans un programme ALGOL 68. C'est la procédure de contrôle (8.1.), interlocuteur de l'utilisateur, qui reconnaît et interprète ces trois symboles.

Le processeur que construit un utilisateur est défini à chaque étape par

- un contexte courant que nous représentons par une chaîne φ
- un état d'exécution.

Le processeur initial est, pour tout utilisateur, le noyau du système, représenté par une chaîne φ_0 , concaténation des prologues standard et de programmation.

Si $\varphi_1 \dots \varphi_i \dots \varphi_p$ est un programme particulier ALGOL 68, son exécution par le processeur ALGOL 68, représenté par φ_0 , résulte de l'exécution du programme

$$\varphi_0 \varphi_1 \dots \varphi_i \dots \varphi_p \varphi_{p+1}$$

où φ_{p+1} représente les épilogues ALGOL 68 [4, 5, 7].

Dans un cadre interactif, on cherche à exécuter un programme particulier par pas successifs. L'action représentée par φ_i est appelée exécution de φ_i par le processeur courant

$$\varphi_0 \varphi_1 \dots \varphi_{i-1}$$

Chaque action φ_i peut entraîner:

- une extension du contexte et une modification de l'état d'exécution (1) et (2),
- une modification seulement de l'état d'exécution (3),
- ou une réduction du contexte, régie par les seules règles de fermeture de bloc, et une modification de l'état d'exécution (4).

Une chaîne φ , représentation du contexte associé au processeur courant, peut contenir des symboles $\{$. Soit $\bar{\varphi}$ la chaîne déduite de φ en remplaçant les symboles $\{$ par le symbole ALGOL 68 $\{$.

Pour que $\varphi_0 \varphi$ représente un contexte courant, il faut que

$$\bar{\varphi} \text{ , } ^n$$

où $\text{ , } ^n$ représente la concaténation de n symboles , , avec $n > 0$, soit un programme particulier ALGOL 68.

Soit $\varphi_0 \varphi$ la chaîne du contexte courant, et n un entier positif. L'utilisateur a le choix entre quatre opérations.

(1) $\psi \#$

où $\bar{\varphi}; \psi)^n$ est un programme particulier ALGOL 68.

Cette opération fait exécuter ψ par $\varphi_0 \bar{\varphi}$, et étend le contexte du processeur qui devient $\varphi_0 \varphi; \psi$.

(2) $\{ \psi \#$

où $\bar{\varphi} (\psi)^{n+1}$ est un programme particulier ALGOL 68.

Cette opération modifie l'état d'exécution et le contexte grâce à l'exécution de (ψ) par $\varphi_0 \bar{\varphi}$. Le contexte courant devient $\varphi_0 \varphi \{ \psi$.

(3) $\{ \psi \} \#$

où $\bar{\varphi} (\psi)^{n+1}$ est un programme particulier ALGOL 68.

Cette opération modifie seulement l'état d'exécution, grâce à l'exécution de (ψ) par $\varphi_0 \bar{\varphi}$. Le contexte courant n'est pas modifié.

Soit $\varphi = \varphi' \{ \varphi''$ tel que φ'' ne contienne pas de symbole $\{$, avec φ' éventuellement vide.

(4) $\psi \} \#$

où $\bar{\varphi} \psi)^{n+1}$ est un programme particulier ALGOL 68.

Cette opération fait exécuter $\psi)$ par $\varphi_0 \bar{\varphi}$ et le contexte courant devient $\varphi_0 \varphi'$.

A chaque étape, l'utilisateur a défini un processeur. On remarque cependant que l'utilisateur ne peut oblitérer (*) aucun accès. Le contexte visible (2.62.) associé au processeur est représenté par la concaténation du contexte visible

(*) sauf par redéfinition d'un identificateur.

du noyau φ_0 (dans lequel certains accès sont oblitérés) et de la totalité du contexte qu'il a construit.

On remarque aussi qu'un processeur ainsi défini n'est pas un objet interne ALGOL 68 puisqu'il contient des identificateurs, et ne forme pas une procédure.

Aussi, on a cherché à isoler une entité ALGOL 68 et à utiliser une opération du langage dont l'application à cette entité fournisse un processeur dans lequel certains accès puissent être oblitérés. Ce faisant, l'expression de modules et de fichiers (2.62.) est rendue possible.

8.41. Le prolator.

L'utilisation d'un processeur est assimilable à l'exécution d'une proposition ALGOL 68 de la forme

$$\begin{aligned} & (\varphi ; \\ & \quad \underline{m_1} E_1 = \psi_1 ; \\ & \quad \underline{m_2} E_2 = \psi_2 ; \\ & \quad \vdots \\ & \quad \underline{m_n} E_n = \psi_n ; \\ & \quad \tau \\ &) \end{aligned}$$

où $n \geq 0$,

. $\psi_1, \dots, \psi_n$ sont des propositions

. $\underline{m_1}, \dots, \underline{m_n}$ sont des modes

. $E_1, \dots, E_n$ sont des identificateurs

. τ ne contient pas d'identificateurs déclarés dans φ . $\tau = \tau_1 ; \tau_2 ; \dots \tau_p$ doit pouvoir être exécuté interactivement par séquence τ_i , $i = 1, \dots, p$.

Exemples: 1) φ est la déclaration d'une pile et d'un index non désignables dans τ

[1:10] ent pile ;
ent hauteur := 0 ;

2) Les déclarations suivantes définissent des objets désignables, du type $\underline{m}_i E_i = \Psi_i$

proc sommet = ent : si hauteur = 0 alors
erreur 1 sinon pile [hauteur] fsi ;
proc empiler = (ent a) : si hauteur = 10 alors
erreur 2 (a) sinon pile [hauteur += 1] := a fsi ;
proc dépiler = : si hauteur ≤ 1 alors erreur 3
sinon hauteur -= 1 fsi ;
proc ent erreur 1 ;
proc (ent) erreur 2 ;
proc erreur 3 ;

3) Les actions suivantes définissent une utilisation du processeur décrit en (1) + (2)

erreur 2 := (ent a) :
 (pour i := 1 pas 1 jusqu'à 10 faire
 (écrire (sommet) ; dépiler) ;
empiler (a)
) ;
erreur 1 := ent : ;
erreur 3 :=: ;
 < suite d'instructions utilisant sommet, empiler et dépiler >

Appelons corps du processeur la séquence ALGOL 68 suivante

(φ ;
 $\underline{m}_1 E_1 = \Psi_1$;
 ⋮
 $\underline{m}_n E_n = \Psi_n$;

Si le corps du processeur peut s'écrire

(I) struct ($\underline{m}_1 a_1$, ... $\underline{m}_n a_n$) x = (φ ; (Ψ_1 , ... Ψ_n))

son utilisation, par l'intermédiaire des identificateurs $E_1, \dots, E_n$ (au choix de l'utilisateur), s'écrit alors

$$\begin{array}{l} \underline{m_1} E_1 = a_1 \underline{de} x ; \\ \vdots \\ \underline{m_n} E_n = a_n \underline{de} x ; \\ \tau \end{array}$$

Cependant, dans (I), les durées de vie des valeurs résultant de l'élaboration de $\psi_1, \dots, \psi_n$ peuvent être limitées à la proposition sérielle dans laquelle elles sont élaborées. Pour éviter cette contrainte, nous introduisons un nouveau genre de proposition sérielle qui hausse la durée de vie des valeurs élaborées à celle de la proposition sérielle immédiatement englobante. Nous appelons prolator une telle proposition sérielle et nous l'écrivons

$$\underline{pro} (\langle \text{expression sérielle} \rangle) .$$

(I) s'écrit alors:

$$(II) \underline{struct} (\underline{m_1} a_1 , \dots \underline{m_n} a_n) x = \underline{pro} (\varphi ; (\psi_1 , \dots \psi_n))$$

Pour donner à l'utilisateur les moyens d'écrire (II) sans avoir à écrire le contenu du prolator, il suffit d'utiliser une procédure p telle que

$$\underline{proc} p = \underline{struct} (\underline{m_1} a_1 , \dots \underline{m_n} a_n) : \underline{pro} (\varphi ; (\psi_1 , \dots \psi_n))$$

et (II) s'écrit alors

$$(III) \underline{struct} (\underline{m_1} a_1 , \dots \underline{m_n} a_n) x = p$$

8.42. Conservation et utilisation d'objets compilés.

Un objet interne ALGOL 68 est caractérisé par son type et par une durée de vie qui définit son domaine d'utilisation. Un tel objet peut être conservé si sa durée de vie est supérieure à celle du programme particulier ALGOL 68 dans lequel il est utilisé. C'est le cas des objets primitifs, de type entier, réel, caractère. C'est aussi le cas d'objets composés d'un type comme

$$\begin{array}{l} [] \underline{ent} \\ \underline{struct} (\underline{ent} a , \underline{bool} b) \end{array}$$

C'est enfin le cas des procédures qui n'utilisent pas d'identificateurs du contexte défini par le programme particulier qui les élabore (voir chap. 7.).

Considérons une procédure utilisant un tel identificateur:

$(\text{ent } y) : x + y$

Comme nous l'indiquions au chapitre 7., il est envisageable de conserver la procédure

$(\text{ent } y, z) : z + y$

dont la durée de vie est universelle, et d'appeler cette procédure avec un second paramètre effectif toujours identique. Cependant, l'introduction d'un nouveau paramètre rend l'appel plus coûteux. Aussi, considérons la procédure suivante:

$\text{proc } p = \text{pro } (\text{ent } x) \text{ proc } (\text{ent}) \text{ ent} : (\text{ent } y) \text{ ent} : x + y$

L'appel d'une telle procédure est équivalent à un prolator de la forme:

$\text{pro } (\text{ent } x = \langle \text{paramètre effectif} \rangle ;$
 $\quad (\text{ent } y) \text{ ent} : x + y)$

qui rend en résultat la procédure souhaitée. p désigne une procédure dont la durée de vie est universelle; elle est conservable.

La conservation d'un objet nécessite, pour y accéder ensuite avec sécurité, la conservation du couple

$(\langle \text{type} \rangle , \langle \text{valeur} \rangle)$

Remarque: Pour des raisons de mise en oeuvre, dues essentiellement au traitement des valeurs unies et au ramasse-miette, on peut avoir à conserver, dans le cas d'une procédure, les types des objets qu'elle utilise.

Le rangement dans un livre d'un objet interne $(\langle \text{type} \rangle , \langle \text{valeur} \rangle)$ nécessite l'emploi d'une procédure spéciale *écrire valeur* de type

$\text{proc } (\text{fichier} , \text{tous les modes})$

où tous les modes est le mode union de tous les types. Cette procédure, déclarée dans le prologue de programmathèque, vérifie que la durée de vie de l'objet est supérieure à celle du programme particulier dans lequel il a été élaboré.

L'accès à un objet conservé dans un livre doit s'accompagner d'un test sur le type de l'objet. ALGOL 68 fournit une telle opération: la conformité. Pour réaliser cet accès, on utilise une procédure spéciale lirevaleur, déclarée dans le prologue de programmathèque, de type

proc (fichier) tous les modes

Exemple: Soit φ une proposition délivrant un objet interne, de type μ et de valeur $v(\varphi)$. Soit f un fichier ouvert sur un livre d'identification "phi".

écrirevaleur (f, φ)

stocke le couple $(\mu, v(\varphi))$.

cas lirevaleur (f) dans

$(\underline{\mu} \ x) : x$

sinon erreur

sac

fait désigner à l'identificateur x l'objet lu si son type est bien μ , et délivre dans ce cas la valeur lue. Dans le cas contraire, il y a exécution de la procédure erreur.

Exemple - Conservation et utilisation d'une procédure.

Soit "pile" l'identification d'un livre ouvert sur le fichier f .

1) conservation d'une procédure.

proc $p =$ struct (proc ent som ,
proc (ent) emp ,
proc dep ,
proc ent e_1 ,
proc (ent) e_2 ,
proc e_3) :

([1:10] ent pile ;

ent hauteur := 0 ;

proc sommet = ent : ... ;

proc empiler = (ent a) : ... ;

proc dépiler = : ... ;

proc ent erreur 1 ;

proc (ent) erreur 2 ;

proc erreur 3 ;


```
(sommet, empiler, dépiler, erreur 1, erreur 2, erreur 3)
);
écrire_valeur (f, p)
```

2) utilisation de la procédure, comme processeur.

Soit un fichier *f*.

```
{ mettre_chaine_valeur ("pile", ... ) } #
```

Cette opération rend le livre "pile" utilisable.

```
{ fichier f ; ouvrir (f, "pile", ...) ;
  mode sixproc = struct (proc ent som, proc (ent) emp,
    proc dep, proc ent e1, proc (ent) e2, proc e3) ;
  sixproc x = cas lire_valeur (f) dans
    (proc sixproc y) : y
    sinon erreur
  sac ;
  proc ent tête_de_pile = som de x ;
  proc (ent) jempile = emp de x ;
  proc jedépile = dep de x ;
  e1 de x := ... ;
  e2 de x := ... ;
  e3 de x := ... #
```

Dans le contexte courant, l'utilisateur dispose de l'objet désigné par *x*, dont il a redésigné certains des champs. Il peut alors utiliser conversationnellement le processeur ainsi obtenu, puis, s'il n'en a plus besoin, réduire son contexte grâce à la commande:

```
} #
```

8.5. CONCLUSIONS.

Grâce au concept de prolator, à l'existence d'un catalogue et des procédures *lire_valeur* et *écrire_valeur*, le système permet la conservation et le partage des objets suivants:

- les procédures utilisant des identificateurs globaux,
- les processeurs ou leurs dérivés, les modules,
- les fichiers au sens d'objets groupant des objets composés et des procédures

d'accès, seules accessibles (*).

En ce qui concerne ces derniers, nous devons noter un certain nombre de problèmes à résoudre. Considérons la procédure p donnée en exemple en 8.42. Pour la considérer comme un fichier, il est nécessaire que la pile soit un objet unique, élaboré une fois pour toutes, et potentiellement partageable. Ainsi l'objet

[1:10] ent pile

doit être remplacé par le texte d'un livre.

On peut alors, tout en gardant à la pile son caractère d'objet partagé, vouloir en limiter l'usage par la seule procédure p .

On doit aussi pouvoir indiquer que la procédure p , partageable par définition, ne doit être utilisée que par un processus à la fois, ou au contraire par plusieurs processus.

Enfin, on doit pouvoir définir deux procédures p_1 et p_2 définissant des fonctions d'accès différentes au livre contenant la pile. Par exemple, p_1 permet d'empiler, p_2 permet de dépiler. Il faut alors limiter l'usage du livre à ces deux procédures seulement. L'usage des livres contenant p_1 et p_2 doit aussi être contrôlé, en fonction des pouvoirs des utilisateurs.

On constate ainsi qu'il faut mettre en oeuvre des dispositifs de protection qui puissent limiter

- d'une part, l'utilisation que fait un utilisateur d'un livre,
- d'autre part, l'utilisation qu'un livre (dans lequel est conservée une procédure) fait d'un autre livre,
- et enfin, le nombre d'utilisations qui sont faites simultanément d'un livre.

(*) Nous conservons évidemment, dans le système, la possibilité de conserver des livres tels qu'ils sont définis en ALGOL 68.

Ces mécanismes étant définis, il est alors très simple de considérer l'outil de communication par boîtes aux lettres comme un prolator délivrant comme valeurs les procédures d'accès à la boîte aux lettres.

Ainsi, la définition complète du noyau passe par la conception de ces mécanismes de protection. C'est, avec la description du fonctionnement parallèle des processus coopérant dans le système, la tâche à laquelle nous nous livrons actuellement.

BIBLIOGRAPHIE

- [1] TRILLING L., VERJUS J.P.

*An Attempted Definition of An Extensible System
in ALGOL 68 Implementation*, J.E.L. PECK (Ed.)
North Holland (1971)

- [2] HARLOUCHET C., RICHER N., VAUCHER J.

Le Système Extensible SAM
Pub. 79 - MONTREAL (Dept-Info) (1972)

- [3] TRILLING L., VERJUS J.P.

Le Système SAR
Note de Travail - RENNES (1973)

- [4] VAN WIJNGAARDEN et al. (Ed.)

Report on The Algorithmic Language ALGOL 68
Spring Verlag - BERLIN (1969)

- [5] BACCHUS P. et al. (Ed.)

Manuel ALGOL 68, par le groupe ALGOL de l'AFCE
A paraître (Hermann)

[6] VERJUS J.P.

Relation avec l'Environnement

Journées ALGOL 68 - Institut de Programmation PARIS (1973)

[7] VAN WIJNGAARDEN et al. (Ed.)

Draft Revised Report on The Algorithmic Language ALGOL 68

Copyright I.F.I.P. (1973)

[8] SHROEDER M.D., SALTZER J.H.

A Hardware Architecture for Implementing Protection Rings

CACM 15,3 (1972)

[9] CROCUS (Ed.)

Les Systèmes d'Exploitation des Calculateurs. Principes de Conception

par un groupe de rédaction

A paraître (Dunod)

[10] TRILLING L., VERJUS J.P.

Le Système SAR

Actes des Séminaires IRIA, *Structure et Programmation des Calculateurs*
(1972)

[11] TRILLING L., VERJUS J.P.

Introduction Dynamique de Processeurs dans un Environnement

Conversationalnel

Article proposé pour publication (1973)

BIBLIOGRAPHIE GENERALE

- ABRIAL J.R.
Structure de Données et de Programmes
Cours. Université de Grenoble (1972)
- ANDRE F., HERMAN D., KERANGUEVEN A., LE PORCHOU M.
Environnement Système pour ALGOL 68
Note Technique 010 - RENNES (1972)
- ANDRE F., HERMAN D.
Choix d'un Mécanisme d'Adressage Segmenté Réalisable sur Calculateur CII 10070 et Adapté à un Système Basé sur ALGOL 68
Note Technique 014 - RENNES (1973)
- ASSABGUI M., MARCIL M., VERJUS J.P.
LP 3100 : un Langage d'Assemblage Evolué
Pub. 42 MONTREAL (Dept-Info) (1970)
- AUROUX A., BELLINO J.
Système en Temps Partagé 1401/7044
Monographie d'Informatique n° 2, DUNOD (1966)
- AUROUX A., BOLLIET L., TASSART G., VERJUS J.P.
Etude Comparative de Nouveaux Matériels Orientés vers la Multi-programmation et le Partage de Temps
Note Technique - IMAG (1966)
- AUROUX A., BELLINO J., BOLLIET L.
DIAMAG : A Multi-Access System for On Line Algol Programming
SJCC Proceedings (1967)
- BACCHUS P., PAIR C., PECCOUD D. (Ed.)
Manuel ALGOL 68, par le groupe Algol de l'AFCET
A paraître (HERMANN)
- BARRON D.W., FRASER A.G.
File Handling at Cambridge University
SJCC (1967)
- BELLOT M., SIRET L., VERJUS J.P.
DIAMAG (II) : Un Système Conversationnel à Accès Multiple
RIRO, 12.B2 (1968)
- BENSOUSSAN A., CLINGEN C.T., DALEY R.C.
The MULTICS Virtual Memory : Concepts and Design
CACM 15,7 (1972)
- BURSTALL R.M., POPPLESTONE R.J.
The POP-2 Reference Manual
Machine Intelligence 2, Oliver-Boyd (1968)

- CLARK D., GRAHAM R., SALTZER J., SCHROEDER M.
The Classroom Information and Computing Service
MAC TR-80 , MIT Press (1971)
- CLEARY J.C.
Process Handling on BURROUGHS B/6500
4th Australian Computer Conference, Adelaide (1969)
- CORBATO F.J., VISSORSKY V.A.
Introduction and Overview of the MULTICS System
FJCC Proceedings (1965)
- CREECH B.
Architecture of the B/6500
Infotech Report 2 (1971)
- CROCUS (Ed.)
Les Systèmes d'Exploitation des Calculateurs. Principes de Conception
par un groupe de rédaction
A paraître (DUNOD)
- GRIFFITHS M., PECCOUD M., PELTIER M.
Incremental Compilation of ALGOL for a Multi-Access System
IMAG (1967)
- GRIFFITHS M.
Systèmes pour la Compilation Incrémentielle de PL/1, ALGOL et FORTRAN
dans *Les Systèmes Conversationnels* , BOLLIET L. (Ed.)
DUNOD (1969)
- GRISWOLD R.
The SNOBOL 4 Programming Language
Prentice Hall (1968)
- HARLOUCHET C., RICHER N., VAUCHER J.
Le Système Extensible SAM
Pub. 79 , MONTREAL (Dept-Info) (1972)
- HAUCK E., DENT B.
BURROUGHS B/6500 - B/6700 Stack Mechanism
SJCC (1968)
- HOARE C.A.R., PERROT R.H.
Operating Systems Techniques
Academic Press (1972)
- HORNING J.J., RANDELL B.
Process Structuring
Comp. Survey 5,1 (1973)

KERANGUEVEN A.

*Implémentation et Gestion sur le Calculateur CII 10070 des Structures
Nécessaires à l'Exécution d'un Système Basé sur ALGOL 68*
Concours ACM - PARIS (1973)

LUCAS P., WALK K.

On The Formal Description of PL/I
Annual Review of Automatic Programming 6,3 (1969)

Mc CARTHY J. et al.

The LISP 1.5 Programmers Manual
M.I.T. Press - CAMBRIDGE (1965)

MORVEN W., MURRAY W.A.

A Proposed Specification of a Conversationnal Computing System for N.P.L.
National Physical Laboratory (1969)

NAUR P. et al.

Revised Report on The Algorithmic Language ALGOL 60
CACM 6,1 (1963)

ORGANICK E., CLEARY J.

A Data Structure Model of The B/6700 Computer System
in Proc. of a Symposium on Data Structures in Programming Languages
SIGPLAN NOTICES (1971)

PAIR C.

Structure de Données
Ecole d'été AFCET (1971)

PAIR C.

Compilation
Ecole d'été AFCET (1972)

PARNAS D.L.

A Technique for Software Module Specifications with examples
CACM 15,5 (1972)

ROSSIENSKY J., TIXIER V.

SAM : The File System
Document Interne - ENSA

SHROEDER M.D., SALTZER J.H.

A Hardware Architecture for Implementation Protection Rings
CACM 15,3 (1972)

- SIRET L., VERJUS J.P.
DIAMAG II
Actes Séminaires IMAG (1968)
- SIRET Y., VERJUS J.P.
Description d'un Compilateur ALGOL Intégré dans un Système à Temps Partagé
Actes Congrès AFIRO (1967)
- TRILLING L., VERJUS J.P.
Définition d'un Système Extensible
Pub. 38 - MONTREAL (Dept-Info) (1970)
- TRILLING L., VERJUS J.P.
An Attempted Definition of An Extensible System
in *ALGOL 68 Implementation* J.E.L. Peck (Ed.)
North Holland (1971)
- TRILLING L., VERJUS J.P.
Notions sur l'Implémentation d'ALGOL 68
Note Technique RENNES (1971)
- TRILLING L., VERJUS J.P.
Le Système SAR
Actes Séminaires IRIA (*Structure et Programmation des Calculateurs*)
(1972)
- TRILLING L.
Traitement des Procédures
Journées ALGOL 68 - Institut Programmation PARIS (1973)
- TRILLING L., VERJUS J.P.
Introduction Dynamique de Processeurs dans un Environnement Conversationnel
Article proposé pour publication
- VAN WIJNGAARDEN, MAILLOUX B.J., PECK J.E.L., KOSTER C.H.A.
Report on The Algorithmic Language ALGOL 68
Springer Verlag - BERLIN (1969)
- VAN WIJNGAARDEN et al. (Ed.)
Draft Revised Report on The Algorithmic Language ALGOL 68
Copyright I.F.I.P. (1973)
- VERGES J.C.
Edition de Textes Basée sur la Reconnaissance Automatique de leurs structures
Assises ACI (1972)
- VERGES J.C., VERJUS J.P.
Reconnaissance Automatique des Structures de Texte en vue de l'Edition
RAIRO B.3 (1973)

VERJUS J.P.

*Description en ALGOL d'un Analyseur-Générateur Conversationnel
pour un Sous-Ensemble d'ALGOL 60
Note Technique - IMAG (1966)*

VERJUS J.P.

*Solutions Adoptées pour l'Ecriture d'un Générateur et d'un
Interpréteur ALGOL Conversationnel
Actes Séminaires IMAG (1967)*

VERJUS J.P.

*Etude et Réalisation d'un Système ALGOL Conversationnel
Thèse Doc-Ing GRENOBLE (1968)*

VERJUS J.P.

*Relation avec l'Environnement
Journées ALGOL 68 - Institut Programmation PARIS (1973)*

VERJUS J.P.

*Quelques Propriétés des Langages d'Utilisation de Systèmes
Ecole d'été AFCET (1973)*

WEGNER P.

*Data Structure Models for Programming Languages
in Proc. of a Symposium on Data Structures in Programming Languages
SIGPLAN Notices (1971)*

INDEX

Accès 2.1
 fonction d'accès 2.1, 2.33, 5.55, 8.23
 chaîne d'accès 2.1

Adresse en mémoire 3.311
Adresse fictive 3.311
Adresse segmentée 3.311
Caractéristiques (d'un objet) 2.1
Catalogue 3.33, 5.1, 8.3
Chainage 3.22, 4.2, 4.3
Chaîne d'accès 2.1
Compilation 4
Conduire 2.1

Conservation 3.1, 3.33, 8.24
 contexte 7.3
 fichier 2.61
 livre 8.24
 objet 3.1, 3.33, 8.24
 procédure 5.54, 7.2
 segment 3.32

Contenu (d'un emplacement) 2.1, 3.2

Contexte 2.52, 6.52, 7.3, 8.1
 primitif 2.52
 particulier 2.53, 6.46, 6.52
 des paramètres 2.53
 visible 2.62

Delivrer 2.1

Descripteur 3.22, 5.2, 6.23
Descriptif 3.313, 5.21, 6.33
Designation 2.2, 4.2, 5.4, 5.5, 6.4, 6.5
Designer 2.1, 2.2
Durée de vie 2.4
Edition de lien 4.3, 5.53, 5.7, 7.2
Elaboration 2.53, 4.4, 5.6, 7.2
Emplacement 2.1, 3.2
Environnement 8.2
Espace adressable 3.32
Etat d'exécution 2.52, 6.44, 7.3, 8.1

Fichier 2.61, 5.53, 5.321, 8.5
 accès 2.61, 5.55, 8.23
 création 2.61, 8.24
 destruction 2.61, 8.24
 fermeture 2.61, 8.22
 gestion 2.61, 8.22
 ouverture 2.61, 8.22
 organisation 2.61, 8.23

fichier (ALGOL 68) 8.22

Fonction d'accès 2.1, 2.33, 5.55, 8.4, 8.5

Formule 2.1

Fournir 2.1

Identificateur 2.1, 2.2

local 2.51, 6.41

global 2.51, 6.41

de paramètre 2.51, 5.45, 6.41

Interprétation 4.2

Lexique 2.52, 6.46

Lien 4.3

Livre 8.2

Machine virtuelle 2.1

Mémoire fictive 3.311, 5.2

Mémoire physique 3.311, 6.2

Mémoire segmentée 3.311, 5.2, 6.2

Module 2.63, 5.542, 8.4

Nom 2.1, 3.2, 5.2, 5.3, 5.4, 6.4, 6.5

Notation 2.1

Objet (interne) 2.1, 2.3

composé 2.1, 2.33, 6.32

élémentaire 2.1, 2.31

global 6.41

inaccessible 2.4

libre 4.3

lié 4.2

local 5.22, 5.44

mort 2.4

primitif 2.1, 2.31, 2.52

Partage 3.1, 3.3, 6.5, 7.2

contexte 6.46, 6.5

fichier 5.321

livre 8.5

objet 3.1, 3.32

procédure 5.1, 5.33, 6.54, 7.2

segment 3.32, 5.2

Préfixe 3.22, 6.21

Procédure 2.1, 2.32, 2.5, 4.5, 5.541, 5.6, 6.33, 6.54, 7.2, 8.4

Processeur 2.62, 8.1, 8.5

Processus 2.5, 5.1, 6.51, 8.2, 8.3

Prolator 8.41

Région 4.3, 5.51, 5.52, 6.43, 6.45, 7.2, 8.23

Relation 2.1, 2.3

conduire 2.1, 2.4

délivrer 2.1, 2.32

désigner 2.1, 2.2

fournir 2.1, 2.33

repérer 2.1, 2.31

Repère 2.1, 2.31

Repérer 2.1, 2.31

Représentation 2.1, 3.1, 3.2

contexte 4.3, 5.6, 6.46, 7.2, 7.3

fichier 5.54

fonction d'accès 3.23, 5.55

identificateur 4.3, 5.4, 5.5, 6.4, 6.5, 7.2, 7.3

lexique 3.33, 4.2, 4.3, 7.3

module 5.542, 8.5

objet 2.1, 3.1, 5.3, 6.3

procédure 4.5, 5.541, 6.33, 6.47, 7.2, 8.42

relation 3.2, 4.3

Segment 3.312, 5.2, 6.2

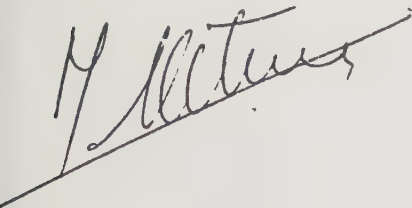
Texte (d'un livre) 8.2

Type (d'un objet) 2.1

Valeur (d'un objet) 2.1

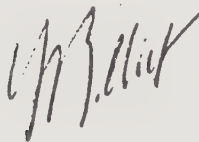
VU :

Le Président de la Thèse



VU :

Le Directeur de Thèse

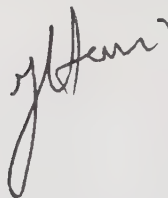


VU ET APPROUVE

RENNES, le 13.07.1973

Le Directeur de l'U.E.R. de Mathématiques
et Informatique

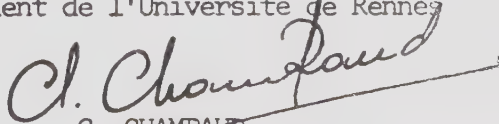
J. HOUDEBINE



VU ET PERMIS D'IMPRIMER N° 125A

RENNES, le 27-8-73

Le Président de l'Université de Rennes


C. CHAMPAUD

